

Adrenaline Junkies and Template Zombies

Understanding Patterns of Project Behavior

项目百态

深入理解软件项目行为模式

- 第19届Jolt大奖获奖作品
- 《人件》作者又一力作
- 入木三分刻画软件项目众生图

[美] Tom DeMarco
Peter Hruschka
Tim Lister
Steve McMenamin
James Robertson
Suzanne Robertson 著

金明 译

人民邮电出版社
POSTS & TELECOM PRESS

Tom DeMarco

软件工程领域权威，软件团队管理圣经《人件》作者，IEEE会士，1986年Warnier奖得主，1999年Stevens奖得主，大西洋系统行会负责人。曾任职贝尔实验室，是结构化分析与设计方法的创始人。

Peter Hruschka

嵌入式实时系统设计和分析专家，用于系统架构文档的ARC42模板开发者，大西洋系统行会负责人。

Tim Lister

软件团队管理圣经《人件》作者，风险管理理论的狂热爱好者。大西洋系统行会负责人。

Steve McMenamin

Hawaiian Electric公司副总裁、CIO，大西洋系统行会负责人，曾任职于Borland等多家知名IT公司，在软件团队管理方面有着丰富的经验。

Suzanne Robertson和James Robertson

均为Volere需求过程和需求分析模板发明人。他们在世界各地举办研讨班，提供咨询服务，帮助大大小小的公司开展需求分析。两人均为大西洋系统行会负责人。

金明　ThoughtWorks高级咨询师，InfoQ资深编辑，SCJP，系统分析师。在企业应用开发领域拥有多年项目开发经验，专注于敏捷原则与方法的理论实践和指导咨询，是敏捷和精益方法与思想的坚定追随者。目前主要关注于大型企业的敏捷咨询、Java/EE、Scala、REST、Web Service以及HTML5，多次在软件开发者社区的活动中发表演讲。译著有《软件开发沉思录——ThoughtWorks文集》、《卓有成效的程序员》等，同时在《程序员》和InfoQ上发表过多篇论文。

图书在版编目（ＣＩＰ）数据

项目百态 : 深入理解软件项目行为模式 / (美) 德马科 (DeMarco,T.) 等著 ; 金明译. -- 北京 : 人民邮电出版社, 2011.3

书名原文: Adrenaline Junkies and Template Zombies: Understanding Patterns of Project Behavior

ISBN 978-7-115-24488-8

Ⅰ. ①项… Ⅱ. ①德… ②金… Ⅲ. ①软件开发－项目管理 Ⅳ. ①TP311.52

中国版本图书馆CIP数据核字(2010)第249868号

内 容 提 要

本书介绍了软件项目行为的 86 个模式，基本上概括了软件项目生命周期的方方面面，揭示了软件项目最常遇到的困境，反省了行业内种种不良习惯和做法。六位作者均来自一个开发咨询的管理团队 Atlantic Systems Guild，长期以来为众多软件公司的经理人提供专业的咨询服务。他们浓缩了成百上千个项目管理的案例，通过本书中一个个模式展现出来。每个模式都以生动形象的插图开始，另外还加上一些趣闻和真实事件。

本书适合所有软件项目的管理者阅读，也适合有志于成为软件管理者的人参考。

项目百态：深入理解软件项目行为模式

◆ 著　　[美] Tom DeMarco　Peter Hruschka　Tim Lister
　　　　Steve McMenamin　James Robertson
　　　　Suzanne Robertson

译　　金　明

责任编辑　朱　巍

执行编辑　郝富强

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号

邮编　100061　　电子函件　315@ptpress.com.cn

网址　http://www.ptpress.com.cn

北京　　　　印刷

◆ 开本：700×1000　1/16

印张：14

字数：233千字　　　　2011年3月第1版

印数：1 – 3 000册　　　　2011年3月北京第1次印刷

著作权合同登记号　图字：01-2009-4833号

ISBN 978-7-115-24488-8

定价：39.00元

读者服务热线：(010)51095186　印装质量热线：(010)67129223

反盗版热线：(010)67171154

对本书的赞誉

“非凡的洞察力。掩上书卷，你会陷入沉思：‘该死，我做了那件事……我们陷入麻烦了！’随即是自我安慰：‘不只是我一个人做了。还有希望！’”

——皮克斯动画工作室软件部副部长 Howard Look

“除了这些作者，谁还能挖掘 150 年的软件团队经验，针对这些时常遇见的场景锤炼出令人过目难忘的名字？我猜测你在自己的工作中会开始使用这些词汇——我已经在用了。”

——《敏捷软件开发》作者 Alistair Cockburn

“对于任何一位曾经在组织里面从事过项目工作的人来说，86 个项目模式熟悉得令人心惊。幸运的是，其中有一些模式是良性的模式，应该给予鼓励。然而悲哀的是，剩下的绝大多数模式不仅仅令人心灰意冷，而且它们对生产率、质量和项目团队士气的破坏程度令人瞠目结舌。”

——《死亡之旅》作者 Ed Yourdon

“作者以高度的幽默感和深刻的洞察力写就此书。本书清楚地讲述了项目因何而失败，有何补救措施。本书以非常友善和令人乐于接受的方式提出了切实可行的建议。”

——哈佛商学院教授 Warren McFarland

“对于每一位经营 IT 组织的人来说，这绝对是一本必读之书。实际上，对于开展各类项目的组织，这本了不起的书里面的教训对其经营者都是适用的。书中的隐喻十分风趣，是那种有点儿笑中含泪、‘你都遇见过’的风趣。你到处都能发现这些常见的症状。携以一定的勇气，手头常常翻阅本书，你定能创造出一个健康的项目环境，使人们在其中能够茁壮成长，同时交付前后高度一致的产品。”

——DTE 能源高级副总裁和 CIO Lynne Ellyn

“人们总是试图去理解自己和其他人。我们的生存依赖于这些理解，生存的质量——从勉为糊口的生计到极为心满意足的生活——也同样依赖于这些理解。人们的个人行为、人际行为及其处在制度坐标下的行为，构成了独一无二的态度和行为框架。针对这些复杂因素的动态变化的思考，铸就了洞察力和影响力。试图理解这些因素的三次尝试跃入了我的脑海。中国人有《易经》，建筑师们有《模式语言》，心理医生有自己的《人类精神失常诊断和统计手册》。本书巧妙地融合了以上三者（在相当程度上吸取了最后一项）的特色，从组织的角度，将人类创建和遵循的模式——与它们的危害和优势——在它们所存在的项目里面关联起来。敏锐、风趣并且一针见血，所有人都应该阅读此书。”

——*The Cluetrain Manifesto* 作者之一 Christopher Locke

译　者　序

什么是项目？按照 PMBOK 的解释，项目是为提供某项独特产品、服务或者成果所做的临时性努力。现代项目管理通常被认为始于 20 世纪 40 年代，但项目管理自古已经开展于一些主要基础设施如埃及金字塔、运河、大桥、教堂、道路、城堡等的建设之中。从古老的“巴别塔”到 20 世纪 40 年代的美国曼哈顿计划，再到如今全球性的项目运作，项目管理的思想和实践已经走过了洪荒时代，进入到成熟、科学的阶段，在各个领域中大放异彩。

然而，相较于其他领域而言，软件还是一门非常年轻的领域。也正因为此，软件自从诞生伊始，尤其是在 20 世纪 60 年代软件危机爆发之后，其从业人员就一直在尝试着借鉴其他领域的思想，引入各种隐喻来描述软件领域的行为和事物，加深人们对于软件开发的理解，提高人们对于软件开发项目的洞察力。《代码大全（第 2 版）》指出：“David Gries 说编写软件是一门科学（a science）（1981 年），而 Donald Knuth 说它是艺术（an art）（1998），Watts Humphrey 则说它是一个过程（a process）（1989），P. J. Plauger 和 Kent Beck 都说它就像是驾驶汽车（driving a car）——可他们两个却几乎得出了完全相反的结论（Plauger 1993，Beck 2000）。Alistair Cockburn 说它是一场游戏（a game）（2002）……而 Fred Brooks 说它像耕田、像捕猎，或像是跟恐龙一起淹死在‘焦油坑’里面（1995）……”

如果拭去隐喻带来的神秘性质，翻开思想和方法的装饰，直面软件领域与其他领域的根本差异，我们可以发现软件的核心特征在于其抽象性。软件开发由需求（问题域）到可执行的源代码（解决域），正是试图以一种抽象的形式捕捉或者展示另一种抽象。软件的抽象性导致了其他领域所不具备的两个重要特征。

- 项目的不可见性。软件开发主要是一项抽象性活动。基于此，计划的执行情况、变更的影响范围、风险的大小等因素对于软件项目都是在一定程度不可见的。如果缺乏对这些因素的把握，项目必然是不可控的，必将走向失败。

- 概念建模的复杂性。软件的本质复杂性源于概念模型的抽象和构建。而这一活动又属于智力的创造性活动，极大地取决于程序员对概念的掌握、理解以及再具象化。这些行动是无法通过灌输、强制等措施来做到的。

这两个特征使得软件开发蒙上了一层诗意的瑰丽色彩。《人月神话》在首篇“焦油坑”中对程序开发有非常形象的描写：“程序员，就像诗人一样，几乎仅仅工作在单纯的思考中。程序员凭空地运用自己的想象，来建造自己的‘城堡’。”软件以及软件开发与其他领域的差异正导致了软件项目异于其他领域项目的一个很重要的特点：对人的依赖性。人是摆在第一位的，正如 Alistair Cockburn 在《敏捷软件开发》之中把软件开发形容为“关于创新和沟通的正和博弈”，能否玩好这个博弈游戏，取决于游戏中每位个体以及他们之间的沟通与协作。如何施以管理，让人发挥最大的创造力，让团队保持持续稳定的开发速度，让项目最终走上成功？

也正因为此，二十年前的《人件》试图从社会学的角度来重新诠释软件的开发和管理，“我们工作的主要问题，与其说是技术性的，不如说更多的是社会性的”，围绕着“高效的项目和团队”探讨了资源管理、环境、团队建设等方面的种种实践。其中，作者指出了“管理工作需要系统思考、启发式判断和建立在经验基础上的直觉”。技术也许可以通过技能训练弥补，但经验的缺失却只能是通过经验的积累和传递。可惜的是，当下的软件管理书籍更多的是来自于学院派的理论和指导，缺少具体经验的传承。比如，项目的不可见性会有哪些形态？如何解决项目的不可见性？如何简化概念建模的复杂性？这些经验对于软件从业人士是亟需且多多益善的。

二十年后，《人件》的两位作者再次携手，延续《人件》的风趣和尖锐，推出了这部《项目百态》。《项目百态》包含了 86 种项目行为的模式，每一种模式都是实际经验的提炼，在风格上与《人件》一脉相承。每一种模式都命以一个通俗直观的名字，配以一张关联的图片和一段点题的说明。大部分的模式都是通过一则有趣的故事来引出问题，再以作者们的真知灼见一针见血地分析问题的根源与特征，最后以作者意味深长的总结收尾。

对于任何一项模式，作者平白表述的背后都是一个个倒下项目的身影。比如“明日复明日”（模式 7），有多少项目因为目标设置过于遥远，导致项目计划对人们不再有紧迫感。又比如“残局游戏”（模式 47），有多少项目沉醉于项目就绪度

舞会之中，却浑然不觉截止日期渐进，而交付物的质量尚不曾经过检验。又比如“牵涉性疼痛”（模式 6），“头痛医头，脚痛医脚”的庸医做法把多少项目扼杀在尚未可期的状态。又比如“主面板”（模式 16），相信有很多项目已经使用主面板来直观了解项目状态，但对于如何设计良好的主面板却所知甚少，作者分享了一些经验。又比如“稻草人”（模式 26），建模是困难的，从无到有的建模更是难上加难，何妨试试“稻草人”，阅后即焚？掩卷沉思，既悚然于种种模式的似曾相识，又庆幸于有些错误仍未出现于项目上，还有时间处理出现的问题。此时，不禁长叹一声：软件项目还可以这么玩！

本书不仅适合于项目团队的各种角色，也适合于软件组织参与到组织间“博弈游戏”的各种角色。事实上，围绕着软件项目的各种角色都能从本书中受益。从项目监控的角度上，分析自身项目的态征，如是否存在“错误的质量关卡”（模式 34）、“造纸厂”（模式 79）等；从团队建设的角度上，团队之中是否存在“英式保姆”，“本”（模式 54），或者“记者”（模式 49）和“影评人”（模式 19），或者勇于跨出“蓝色区域”（模式 44）的人；从项目交付上，团队是否交付了“没人在意的交付物”（模式 61）；从项目生态管理上，“空椅子”（模式 50）、“堆积”（模式 77）是否存在于项目之中，各种角色是否“互相教学”（模式 65）。

当然，相对于《人件》，这本书可能在结构上存在着白璧微瑕。它没有像《人件》一样把相似的模式归并在一起，而是选择了以相对零乱的顺序来阐述。事实上，本书中的很多模式，比如“错误的质量关卡”、“造纸厂”等都可以归为一类。但这或许也是作者的匠心，并不试图勾勒软件项目的全貌或者结构，而是试图从不同的视角来观察、投影软件项目，然后从每一个视角窥一斑而识全豹。回忆对未知事物的认知模式，分解和投影往往能给人相对直接和较深入的体验。这些体验汇合在一起，给我们一个对软件项目的全面而直观的认识，又岂不快哉？

本书的翻译得到了很多人的帮助，我无法逐一列出。首先，要感谢刘江，正是他的推荐，才让我没有与这本书擦肩而过。其次，要感谢图灵的各位编辑们：他们的耐心，让我可以慢慢地斟酌自己的翻译；他们的细心，帮我发现了不少错误；他们的专业，让本书的排版和标注不再凌乱。

感谢我的父亲金建华和母亲胡赛花，我对文字的热爱很大一部分是来自于父亲深夜挑灯读书的背影，而母亲一直尊重我对软件工作的选择，即使那意味着她的儿子离开自己、置身于千里之外。虽然他们都已经不在了，但是当我孤单的时

候，他们的目光一直是我前进的动力。我要感谢我的弟弟金成，他承担了照顾父亲和母亲的大部分责任，让做兄长的我可以将更多的精力放在工作上面。我还要感谢我的女朋友，虽然我们身处两地，但她始终给予了我足够的信任和支持，让我可以做我感兴趣的事情。

最后，由于本书是由六位风格迥异的作者写就，各成风格却又风趣一致。将原意表达出来又不失趣味是我的目标。虽然我已经下了不少功夫，但终究经验和能力有限，译稿之中难免有疏漏之处，欢迎读者批评指正。

实话的力量

我对我的客户、一支百人团队的领导说："你们并不是没有优先级。你们的优先级策略是最后来的事情优先。"

他苦笑不语。因为他知道，这就等于在说，他手下的百十来号人基本是在做布朗运动；更因为他知道，这是实话。只是他自己不能说，他的手下人也不能说。

事情经常是这样：尽管所有人都知道，但谁也不会把它说出口，因为说那样的话是政治不正确的。比如说吧，也许你早就知道你手上那个项目注定是条死鱼，但你敢说出口吗？更多时候，你会让自己变得乐观——乐观程度与最后期限的紧迫程度成正比，直到时间夺走你所有的手牌。

另一些时候，你知道自己碰巧做对了某些事，你不愿意新来的经理改变它：取消每周四晚上的三国杀，把团队从大会议室里搬回格子间以便"释放资源"，把几个模块外包到一千公里外的另一个城市。可你没办法说服领导。"你有度量证据吗？"当然，你没有，而削减成本总是政治正确的。

情况不会自己变好的，如果人们连真实情况都不敢说出来的话。

幸运的是，像 Tom DeMarco 这样名声和年纪（这很重要）都足够大的人可以不在乎政治正确性——换句话说，他们可以说实话。这本《项目百态》就是他们关于项目管理的实话集。

在这本实话集里，作者们挑选了 86 个项目管理中常见的模式，从"肾上腺素成瘾"[①]直到"模板僵尸"——从英文书名不难看出，这无非是"从 A 到 Z"的又一个变体。这些模式，诚如一位评论者所说，有良性的，更多的不仅恶性，而且丑陋可笑。读这样一本书，你会笑，更多的时候你会摇头苦笑，甚至如芒在背。"我的团队没有肾上腺素成瘾吗？"很多读者将很难面对这个问题。

那就对了。

---

① 这是原书中第一个模式名，本书译为"玩的就是心跳"。——编者注

我要感谢我的同事金明向我推荐这本书：几位作者帮我们消解了政治正确性的风险，我们就可以放心地说出实话，然后努力进步。下次当同样的模式出现时，我就可以大声地说："这就是《项目百态》中的××模式。"或者当我再次面对前面提到的那位客户时，我可以把书递给他，对他说："也许你该看看××模式。"

也许你也应该看看。

熊节

ThoughtWorks 中国公司首席咨询师

《重构》译者

照片授权说明

引言　版权所有© 2007 Peter Angelo Simon (www.PeterAngeloSimon.com)
模式 1　Kelley Garner 作品，由 iStockPhoto 提供
模式 2　版权所有© Byron W. Moore，由 BigStockPhoto 提供
模式 3　版权所有© Stefanie Timmermann/iStockPhoto
模式 4　图片由 Chris Linn 授权，使用得到 Corporate Entertainer (www. chrislinn.com) 许可
模式 5　版权所有© Darryl Mason/iStockPhoto
模式 6　版权所有© 2005 Tari Faris/iStockPhoto
模式 7　Mark Lisao 拍摄 (markldxb@gmail.com)
模式 8　版权所有© 2006 Leah-Anne Thompson/iStockPhoto
模式 9　情绪戒指 Bruce MacEvoy 作品(www.handprint.com)
模式 10　版权所有© Joseph Jean Rolland Dubé/iStockPhoto
模式 11　Rembrandt's “Faust” 摘自 Wikipedia Commons
模式 12　作品 llwill/iStockPhoto 作品
模式 13　gocosmonaut/iStockPhoto 作品
模式 14　版权所有© 2007 Lise Gagne/iStockPhoto
模式 15　版权所有 Aleksandr Ugorenkov/iStockPhoto
模式 16　图表由 James Robertson 绘制
模式 17　*Academe: Faculty Meeting*，版画作品，3.9 英寸 × 6.1 英寸（100mm × 155mm），©2002 Evan Lindquist-VAGA/NY
模式 18　Timothy Lister 作品
模式 19　Dan Leap/iStockPhoto 作品
模式 20　版权所有©Tim Pannell/Corbis
Interlude，“Project-Speak”：版权所有© Alexei Garev，使用得到许可
模式 21　版权所有© 2007 Milan Ilnyckyj (sindark.com)
模式 22　Tom DeMarco 作品
模式 23　Zennie/iStockPhoto 作品
模式 24　David Lee | Agency 作品 Dreamstime.com
模式 25　版权所有 Karen Squires/iStockPhoto
模式 26　Suzanne Robertson 作品
模式 28　Emin Ozkan/iStockPhoto 作品
模式 29　由 Images of American Political History 提供
模式 31　© Nicemonkey | Dreamstime.com
模式 32　版权所有© Maciej Laska/iStockPhoto
模式 33　“玩扑克的人”Pro Hart 作品，使用到得许可

模式 34　Kativ/iStockPhoto 作品
模式 35　© Undy | Dreamstime.com
模式 36　版权所有© Bruce Lonngren/iStockPhoto
模式 37　Scott Olson 作品，版权所有© 2007 Getty Images
模式 38　版权所有 Roberto A. Sanchez/iStockPhoto
模式 39　版权所有© Nick Martucci | Agency: Dreamstime.com
模式 40　版权所有© Tim Davis/CORBIS
模式 41　版权所有© 2007 Felix Möckel/iStockPhoto
模式 42　“The Atlantic Trench”，摘自 Wikipedia Commons
模式 43　版权所有© Frances Twitty/iStockPhoto
模式 44　John T. Daniels 作品，Library of Congress/Wikipedia Commons
模式 45　按从左到右的顺序，(a) 版权所有 Maartje van Caspel/iStockPhoto；(b) 版权所有 Guillermo Perales Gonzales/iStockPhoto；(c) 版权所有 Michael Kemter/iStockPhoto；(d) 版权所有 Duncan Walker/iStockPhoto
模式 46　“Untitled 12”(2001) Tai-Shan Schierenberg 作品，由 Flowers，London 授权
模式 47　bluestocking/iStockPhoto 作品
模式 48　由 Borys Stokowski 授权使用作品
模式 49　doodlemachine/iStockPhoto 作品
模式 50　James Robertson 作品
模式 51　20TH Century Fox/The Kobal Collection
模式 52　Yails/iStockPhoto 作品
模式 54　版权所有 Pathathai Chungyam/iStockPhoto
模式 55　版权所有© 2007 Don Bayley
模式 56　版权所有© 2006 Joanne Green/iStockPhoto
模式 57　版权所有© 2005 Rob Friedman
模式 58　“Confrontation 2”，Chaim Koppelman 和 Terrain Gallery 作品，使用得到许可
模式 59　版权所有© eyeidea/iStockPhoto
模式 60　版权所有© Sawayasu/iStockPhoto
模式 61　版权所有© James Rye
模式 62　版权所有 2007 Michael Altschul/visuelmedie.dk
模式 64　照片由 Marja Flick-Buijs 授权，使用得到许可
模式 65　版权所有 Dmitriy Shironosov/iStockPhoto
模式 66　TOHO/The Kobal Collection
模式 67　照片摘自 Wikipedia Commons
模式 68　Gabriel Bulla/Stockxpert 作品
模式 69　CBS/MCA/Universal/The Kobal Collection
Interlude，“The Cutting Room Floor”：版权所有© Valerie Loisdeux/iStockPhoto
模式 71　木版画，Joseph Taylor 作品，版权所有© 2007
模式 72　照片由 Weir Valves & Controls 英国公司提供
模式 73　由 Kunsthistorisches Museum、Wien oder KHM、Vienna 提供
模式 74　版权所有 Pattie Calfy/iStockPhoto

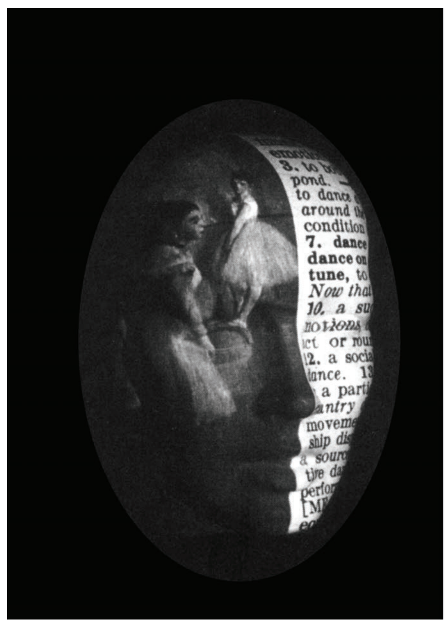
模式 75　James Rye/BigStockPhoto 作品
模式 76　chuwy/iStockPhoto 作品
模式 77　版权所有© Radius Images/Alamy
模式 78　“Samson and Delilah”木版画，Bobby Donovan 作品，版权所有© 2006
模式 79　版权所有 Tim Messick/iStockPhoto
模式 80　Peter Hruschka 作品
模式 81　Peter Hruschka 作品
模式 82　版权所有 Yanik Chauvin/iStockPhoto
模式 83　Brian Duey 作品（www.dueysdrawings.com），版权所有© 2006，使用得到许可
模式 84　使用由 James Dyson 授权
模式 85　版权所有© John Carleton，由 Dreamstime.com，德版版权所有 Lise Gagne

版 权 声 明

引　　言

抽象为人类独有。我们现在每天、每个清醒的时间都会进行抽象，但曾经却并非如此。在史前时代的某个时刻，抽象可能是这样萌芽的：一个早期智人凝视着一样依稀很熟悉的东西，他的脑海中闪过一个想法：“嗨！又是那个东西！”这就是最原始的抽象。从那时起，一切都不同了。人类在地球上不再受自然羁绊。

抽象完全是人类的行为。模式识别则不然，它并不为人类所独有。老鼠很清楚猫儿何时进入梦乡，人们何时不在厨房，面包屑又是何时掉落尚未被清扫。兴许你只是突发奇想，打算离家待一个周末，你的家犬也能提前察觉你的意图。（是不是因为你的手提箱？）而你的邻居浣熊也知道当潮水退去，遗留在海滩上的东西必然比你肥料堆里的东西要好得多。然而，虽然熟稔于模式识别，老鼠、狗、浣熊却不会观察，联想不到“嗨！又是那个东西”。这其中就需要抽象了。



模式识别和抽象的区别在于它们捕获本质的方式。模式随着时间被吸取和提炼，保留在脑海的深处，很难用语言来表述，而多半是以直觉的方式传达给你的。直觉，比如觉得持球队员将向左边冲击或者配偶的怒火即将爆发，都是源自过去识别出的模式。至于这周项目状况的会议上将引发争论的直觉，也不例外。不能明确表述的模式并非毫无价值——毕竟存在即合理，但如果你仔细考量，并且基于它可表述的观察结果进行探究，模式的价值就能显著地提升。

举例来说，问自己这个问题：过去的一年中，引发争论的会议之间存在哪些

共同之处？呃，这些会议大部分都是这样的：老板的老板也参与了会议，而且通常是季度末期的会议。最糟糕的会议则当属——团队在会上汇报了计划表的失误。据此，你能形成一个模式：“我的老板对会上汇报的失误简直会发狂，特别是在接近季度末期的会议上面，因为他的老板那时也会出席。”

尽管已识别的信号可以推导出这个观察结果，但是它们仍然隐藏在无意识之中，偶尔才能让你产生直觉。然而，在右脑直觉和左脑表述能力交汇的一刹那，你抓住了本质，并且转化成清晰的语言。你可以把它写下来，使用明确的测试校验其正确性，和其他人分享，与一同工作的人合并各自的观察结果。

大部分做项目管理工作的人都非常善于识别模式和培养直觉（“我感觉这个项目会走向灾难”），但却不那么善于把模式抽象成更加有用的格式。我们几位作者一起研讨，从加起来长达 150 年的相关从业经验中吸取出特定的模式，再精确地表述出来——于是有了这本书。

一本书的格式会影响到内容陈述的顺序，因为必然是一页接着一页才能成书。但模式本身并不具有天然顺序。我们按照自己的喜好排列这些模式，从第一页到最后一页，争取给读者带来最愉悦的阅读体验。

无论你是从头读到尾，还是随意翻阅，脑海中请始终记住这一点：我们并没有宣称自己观察到的模式具有普适性。显然，它们不可能是到处适用的。别人给出的模式也许适合你的组织，也许不适合。如果适合，我们希望能帮助你将曾经只是直觉的东西转化成观察结果，你和你的团队可以表述、验证和提炼它。

在本书的创作过程中，我们从建筑大师、哲学家 Christopher Alexander 和他令人惊叹的《模式语言》[①]一书那里受益匪浅。在那本影响深远的著作中，Alexander 与合著者明确指出了建筑的几百种模式。该书帮助我们更好地理解我们所涉及以及想涉及的部分，而且，该书也向我们展示了一点，即经过深思熟虑的清晰抽象足以阐明任何事物。

① C.Alexander 等人所著的 *A Pattern Language : Towns, Buildings, Construction*（纽约：牛津大学出版社，1977 年）。

目　　录

模式 1　玩的就是心跳



组织相信忙乱的工作象征着高效的生产率。

电话响了。

“我们想这周完成需求的规格说明书。你能过来看看可以做些什么吗？”

“规格说明书怎么了？”

“我们很急，所以新招了许多人来撰写规格说明书。我们觉得他们完全不清楚自己在做什么。”

“如果由我们来指导他们编写需求，效率不是更高吗？”

“但是我们这周就需要规格说明书。”

“好吧，我明天过来。”

两个小时之后。

“你能过来看看我们评估的工作量吗？”

“规格说明书怎么办？”

“我们没时间了。我们就照现有的需求规格说明书进行。老板要求今天就把评估的工作量交上去……”

你可能已经识别出这一类“玩的就是心跳”（adrenaline junkie）型组织的特点了：优先级总是变化不休，所有事项都是“昨天”就要，总是没有足够的时间交付项目，所有项目都是紧迫项目，紧迫的项目源源不断。每个人都忙得焦头烂额……永远如此。

这些组织里面的人不会从战略层次上思考问题，只是按照紧迫程度来完成工作。除非“忙乱指数”非常高，否则它通常都会被忽略——尽管它很有可能会带来显著的长期优势。人们会一直对其不闻不问，直到它突然（绝对出乎意料地）变得非常紧迫。“玩的就是心跳”分子相信最好的工作方式不是先谋而后动，而是竭力追赶时间。

这种组织文化认为紧迫性越高的项目，绩效也越高。如果身处这种文化之中，你很难不受感染：紧迫性是受到鼓励的。那些为了某个短得可笑的期限而加班到深夜的程序员被视为英雄（根本不在乎他们交付的程序质量）。每个周末，团队的所有成员都照常上班，以保持他们的工作量：这样做的团队比不这样做的团队更受人赞许。此外，如果你不是一直都过度加班，或者没有发疯一般忙碌，你就会被贴上“不是自己人”的标签，你不是保障组织运转流畅的大忙人之一。非英雄行为绝对不能被接受。

绝大部分的“玩的就是心跳”型组织至少存在一个瓶颈，就是那位英雄。他包揽了所有设计的决定，是全部需求的唯一来源，或者一个人决定框架的方方面面。他扮演了两个角色：一是让自己表现得比常人难以想象的还要忙；二是引发决策僵局，他的决策一旦公布，就会导致组织的其他部分更加忙乱。

大部分的“玩的就是心跳”型组织满腔热情地拥护客户服务的理念：他们错误地认为对紧迫事件的响应就是值得赞赏的响应。当客户提出了一个请求，不管是否存在潜在的利润（甚至有效性），该项请求都会立即转化成一个项目，而且通常截止日期会短得可笑。（更多讨论，请参阅第 38 项模式。）这个新项目自然会加重本已超负荷工作的英雄们的负担，使他们更加手忙脚乱——所有的这一切无休止地让组织变得非常、非常忙碌。很多这一类的组织都（错误地）认为这就是敏

捷的全部。

“玩的就是心跳”型组织经常会断然行动，而不是三思而后行。这样导致的结果就是大部分工作都处在不断变化、无法固定的状态，没有什么可以固定下来，或者保持较长的时间。这种不固定的状态一直延续：需求规范不固定——没人真正清楚要构建什么；设计和计划也不固定——它们很可能明天就会改变。紧迫性是唯一的标准，没有人尝试按照重要性或者工作价值来对工作排定优先级。

对于“玩的就是心跳”型组织，回春妙手是不存在的。他们完全是没救了，除非消除那些令人心跳加速的事情，而且解雇经理，雇用那些明白“组织只有不再忙于处理突发事件才最有效率”的人。但是这样的人事变动根本不可能被采纳，因为高层领导，通常是 CEO，希望看到组织长久地保持匆忙的状态，因为工作匆忙让人生出一种高生产率的幻觉。而且，如果公司的经理们是“玩的就是心跳”分子，项目团队也不会相去太远。

“玩的就是心跳”型组织并不是总会失败，他们当中有一些在多年里一直保持着匆忙的节奏。但是，它们都不可能构建重大的东西——那需要稳定性和计划。亢奋型行为不可能扩展——由一些相对较少的人在没有方向，也没有战略指导的前提下，光凭借非常、非常忙碌的工作所能达到的效果十分有限。

当然了，任何组织内都会遇上紧迫的事情，也需要有一些角色去关心紧迫的任务。但是，不是所有的事情都是紧迫的，也不是所有的角色都要关心紧迫的任务。除非化紧迫性为优先级和约束，否则，这种“玩的就是心跳”的治愈希望是微乎其微了。

模式 2　快，赶上



当项目团队决定谁在何时该做什么事情时，呈现出明显的紧迫感，并迫不及待地想立即采取所有必要的行动。

想象你是一只苍蝇正停在墙上，此时开发团队正在召开例会。随着会议的进行，你能看到下面的角色分工，听到下面的交谈。*我们要解决的问题是什么？解决方案的要点是什么？每项要点由谁负责？首先需要完成的事情有什么？谁来完成它们？在什么时间之内完成？如果我们不清楚某一项个别的任务要花多长时间，谁来定义任务范围，以及何时才能完成？什么时候我们再聚一起，计划接下来的任务？*好了，散会。

会议之后，通常在一个小时之内，所有成员会收到一封邮件，上面总结了团队达成的行动计划。事实上，往往在这封总结邮件发出之前，其中一项或多项行动事项就已经完成了。人们在会议结束之后就立刻投入到行动之中。

即使在墙上，以你苍蝇的眼睛也能看出来这是一支高效的团队。

我们知道对于一支高效的团队而言，人们在会上就开始处理商定的事项，这一点都不稀奇。通常，接受任务的人会认为相较于把任务写下来会后再做，现在就做反而更容易一些——比如“把所有尚未解决的优先级为 2 的缺陷分派给产品管理工程师进行分类”这个任务。如果只有询问过会议之外的某人之后才能做出

决定，那么接受任务的人就会给那人发送即时消息，并且把结果汇报给与会人员，保证行动计划的制订工作可以继续进行。

这种“立刻行动”的例子非常特殊，而且应当承认它们是拜技术所赐。但是行动的内在基础来源于团队的文化，而不是所使用的技术。分派缺陷到底是花费 10 分钟抑或 90 分钟并不是重点，重点是团队立刻就着手开始。在团队日常“快，赶上”的活动中，你能观察到下面一些特点。

- 他们对于时间的紧迫性有着内在的直觉。他们把延迟视为成功的真正风险，不需要用截止时间来督促。他们竭尽全力，一旦完成，就把产品推向市场（或者把系统做成产品）。他们明白时间就是金钱。
- 他们对个人和集体的能力非常有信心。想象一下赤足走在一间黑暗且完全陌生的房间里面。当你不确定前面有什么东西，或者不知道下一步会撞上什么东西的时候，你会慢下来。信心不足就像一种摩擦阻力。以行动为导向的团队对自己决定和行动的正确性（或者可纠正性）非常有信心，所以他们一往无前，勇无所惧。
- 他们相信迭代的价值。他们不会特别担心做错了事情——部分是因为他们很自信，但同时也因为他们完全准备好了随着工作的进行，经常做出评估，不断修正方向。摆脱掉“每次决定都必须毫无瑕疵”的包袱，但同时坚信大多数决定都是正确的，他们可以很痛快地做出决定，并付诸行动。

也许值得花一些功夫考虑一下相反的模式，“脱口秀”会议——它们有时十分有趣，但却缺乏行动。这种会议有如下几种形式。

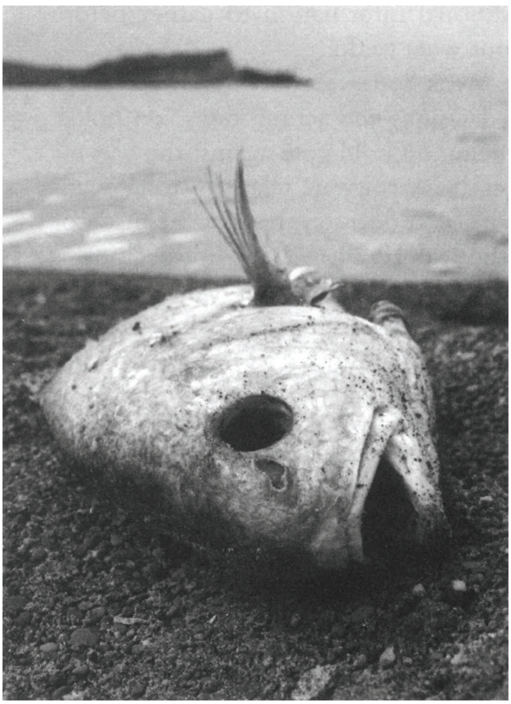
- 要求完美的信息。有些公司的文化把不犯错误看得比完成事情还重。换句话说，什么也不做都比做错了更安全。这种文化催生了这样一类领导和团队，他们企图搜寻到所有的信息，在第一次就做出绝对正确的决定。通常，团队会议的结果不是决定出做什么，而是决定出还需要收集哪些额外的信息以决定做什么。
- 对“等一等”的膜拜。弱团队比强团队更倾向于推迟他们的决定和行动事项。“推迟”的理念与强软件团队的行动方针背道而驰。强团队渴望把事情全部做完。如果需要做出决定，或者决定完成一项任务，团队一定会去完成。如果决定被合理地推迟到未来的某个时间，团队会为其在开发活动中安排一个特定的点。弱团队则总是会寻找借口，等着晚一会儿再做决定或

者采取行动。

- 未定事项的大浆糊。会议组织不善，一个接一个的新想法冒出，从一个主题跳跃到另一个主题，一个接一个的议题被提出，但却没有一个能够得出结论。
- 篝火旁边的轶闻。一些团队会议毫无组织，纯粹是一系列的轶事趣谈，以及对组织过去和现在的口头传说的回忆。
- 条条大路通设计。当团队由架构师和开发人员主宰时，我们有时观察到，无论会议的目标和议程如何，每一次会议都最终演化成对设计的讨论。设计讨论也许很精彩，但如果妨碍了其他更有价值的议题，就不再如此了。
- 开会安排额外的会议。所有失败的会议最终都走上这一步，概莫能外。

模式 3　死鱼

自打开工起，项目就完全不可能完成目标，项目团队中的大多数人都很清楚这一点，但却都缄口不言。



很多IT项目的目标可以被扼要地概括为：我们要在截止日期之前，完成这一系列功能，达到这种准确度，并且保证足够的健壮性。项目团队于是组建起来，项目目标和约束条件转化成详细的需求和设计，并且公布于众。

一个天大的秘密是项目团队中没有一个人相信项目最终能成功。通常看来，如果其他目标不做修改，截止日期是无法达到的。不可思议的是，没有人指出失败的阴影正如一只散发恶臭的大死鱼一样把项目变得臭不可闻。

希腊式悲剧拉开帷幕，项目进行得步履维艰。然后一如既往地，在预期交付日期之前的几周，所有的项目成员、项目经理、项目经理的上司以及任何与项目沾点边的人，要么——

(1) 对于项目没有按照即将到来的发布日期进行，表示震惊、气馁、诧异；要么——

(2) 保持低调，除非被点名问到，否则在任何事情上都一言不发。

为什么如此多的组织里面有如此多的人给实际情况喷洒上除臭剂，而不是简单地指出“项目现在的方式不是我们所希望的方式，死鱼就在那儿”呢？

很多组织是如此汲汲于成功，以至于任何表达疑虑的人不会因为讲出由衷的意见得到任何奖赏。事实上，如果谁在项目的前期阶段就声称死鱼的存在，管理

层的第一反应多半如下。

> “证明给我们看。给我们证明成功的可能性只是零。不要拿以前项目的死鱼经验来唬人。现在的项目不一样。用严密的数学证明来告诉我们失败无法避免。”

一旦你提出任何缺乏精确证据的东西，就会被指责为软蛋或者是试图逃避朴实的辛勤工作：

> “你是软蛋，还是懒汉？你自己选择吧。但是我们不相信你还能在这个良好的组织里面待很长时间。”

在这样的环境里面，“努力”而无法完成比站出来指出目标无法达到更安全。确实，有时需要勇于承担有挑战性的项目，在认输之前真正地拼搏一番。非常正确——但区别是，在有确定截止时间的艰难项目中，没人会捱到最后一刻才声明情况的危急。如果项目是给只有 18 个月就要上天的通信卫星构建软件（你知道如果错过了这次发射时间，下次机会就要再等 16 个月），那么，你们每个人每天都要闻一闻项目有没有出现什么异味。你非常清楚“死鱼”项目只有到后无退路的情况下才会采取行动，所以，一旦闻到异味，你就会立即采取行动。

非常清楚的是，“死鱼”不仅给组织带来破坏影响，而且打击了“死鱼”项目团队和他们经理的士气。无论组织文化如何，没有人会觉得长期作为发臭的“死鱼”项目成员会很舒服。严密捂盘“死鱼”消息的成本实在太高。

> 献给《蒙特・派森》（*Monty Python*）迷们：
> “项目还没死，它正在附住峡湾！”
> “项目没有死，只是在蜕皮！”
> “这项目已死。它已经驾鹤西去了。”
> “现在上场的是完全不同的模式……”

模式 4　欢乐的鼓掌会议

是否表现出高涨的士气成为个人绩效的评价因素。



Chris Linn, Corporate Entertainer(www.chrislinn.com)

高涨的士气永远是组织健康运转的一个象征。与之类似，低靡的士气则说明肯定有什么地方做错了。有一种管理理念就是奉这种关系如圭臬，试图从相反的方向来利用这种关系。这种理念的逻辑是这样的：把士气鼓舞起来，其他美好的东西也就接踵而至。

啊哈，那如何鼓舞士气？尤其是如何在不增加时间、精力和花销——虽然它们是改善事物的必要投入——的情况下鼓舞士气？这是个难题，但别以为人们就不会去尝试。于是，才有了这样的谚语："打气，打到士气提升为止。"

鼓舞士气的常见举措是仪式性的会议。会上老板笑容可掬，站在集合好的团队前面，大开言路。"让我听听你们的心底话，"他大度地告诉大家，"任何事情都行，坏的消息和尖锐的问题也可以。"注意这里的腔调和背后传达的信息：没有什么好隐瞒的，我们是一个欢乐的大家庭。（欢乐，该死的，欢乐。注意了。）

我知道有一家公司，这样欢乐的鼓掌仪式被称作全体会议。之所以

> 称为全体会议，是因为所有人都获邀参加。但是一旦哪位勇敢的人果真举起手，向 CEO 提出尖锐的问题，最后的结果却根本不是他所预想的。CEO 咕哝了几句，就迅速改变话题，不再讨论这个问题。在当天的晚些时候，这个冒失的提问者就被他的顶头上司叫去训了一顿，本来还指望这类尖锐问题会很受欢迎呢，结果这种幻想彻底破灭了。从此以后，在私底下大家都把这种会议都称为一言堂，因为没人会再次举手发言了。
>
> ——TDM

一旦理解到老板并不是在征求你们的发言，而只是征求你们的同意时，你就能清楚地知道要上演什么好戏了：欢迎参加欢乐的鼓掌会议。

模式 5　保姆型项目经理



项目经理拥有的技能与传统的英式保姆有很多共同之处。

一个优秀的项目经理要对手下员工的能力了如指掌。他分派任务、制订计划，在可用的技能和任务本身的要求之间寻求最佳的契合点。这是显而易见的。还有一些项目经理更进一步：他们提供一个工作环境——不仅是技术性的，而且是社会性的——让人们可以最大限度地使用自己的技能，并且提高这些技能。这些项目经理确保自己的员工拥有完成任务所必需的工具。这些项目经理鼓励提问，并且乐意与员工辩论；他们给每位团队成员设定最合适的挑战；他们在需要的时候提出批评；他们建设一个人人乐于工作的场所；而且他们采取必要的调整以保障各项事宜运作良好。简单地说，好的经理*培养*他们的员工，就像保姆培养她们照看的孩子一样。

一个*保姆*，在传统的英式文化中，受雇于某个家庭来照看孩子。保姆通常要具有教师、护士和厨师的技能，对孩子的体格、心灵、社交、创造性和智力发展都要负责。在每天的日常活动中，保姆要确保孩子远离伤害，保证孩子们得到了足够的新鲜空气与锻炼，食用有营养的食物，并且增进对世界的理解，学会更多在世界上生存的技巧。除了照看孩子，保姆还需要与孩子的父母就孩子成长方面的顾虑进行沟通，鼓励孩子的特殊天赋。保姆创建出一个安全的环境，使孩子能

适当冒险并从中学到新知。

经理们拥有这些与保姆类似的才能，就能通过鼓励和培养员工的天赋，从员工那里获得更多、更好的工作成果。

> 迄今为止，我为之工作过的最好的经理是 Peter Ford。这再明显不过了，比如他让员工在工作时各尽其能。举一个例子，我们在一大间开放式设计办公室工作，这不是最好的思考环境，于是他设法为团队弄到一些消音的屏幕，并保留了几间“静室”。所有的这些，还有他为我们做的其他事情，牵涉了我们所不知情的协商和政治。他鼓励我们阅读，在系统开发的时候讨论新的想法。他为团队购买书籍和杂志，并安排时间让我们聚在一起讨论。当我们觉得不开心或者不舒服的时候，他会注意到这一点，跟我们交谈，帮助我们。他保护我们不受组织其他人和事的干扰，但如果他对我们不满意，他会让我们知道。他办公室的门很少关着。Peter 就是我们的保姆。
>
> ——SQR

如果注意到如下一项或者多项情况，那么你所在的组织就已经有一些“保姆型”经理。不必预约就能见到你的头儿，或者不必在琐碎和令人生厌的管理工作上花费太多时间。周围是开放的氛围，人们畅所欲言，互相学习。这种经理认为培训或进修非常必要，而不是视之为奢侈烧钱。他们还会单独抽出时间（比如早晨咖啡闲谈或者周五下午阅读探讨），让大家在一起讨论新的想法。

在任何由人组成的团体里面，总会有流言蜚语、小道八卦和一些磨洋工的情况。然而，在被经理细心呵护的办公室里面，这类浪费时间的事情会极少，因为经理确保团队成员对于实际进行中的事情都非常清楚。人们不需要去靠打探小道消息以得知组织内发生的事情。与之相反，他们觉得自己充分知情和受到信任，把精力都放在自己的本职工作上面。

一个类似于保姆的经理会把他自己看成是工作的催化剂。传统保姆的工作满意度来自于看到孩子能力的发展，“保姆”型经理的工作满意度则来自于看到每个团队成员在个人角色方面得到发展、生产率得到提高，并对自己的工作更加满意。

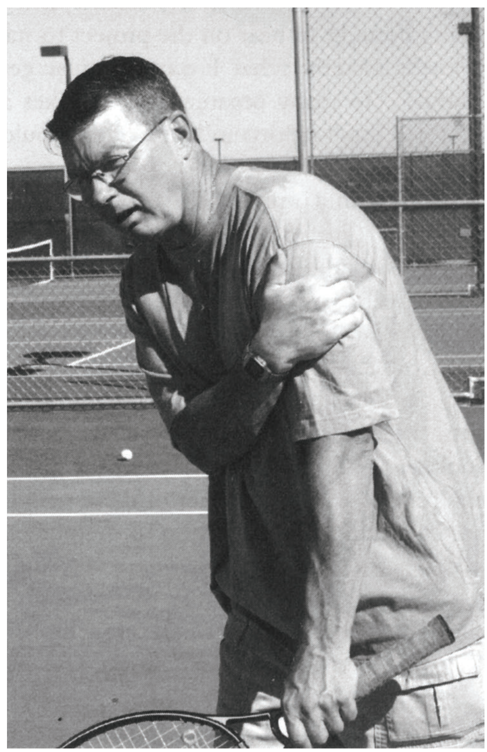
这个模式的反模式有：经理把工作重心放在员工内部的明争暗斗上面、放在行政管理上面、放在流程上面，抑或放在逢迎更上层领导上面；绘制、调整 PERT 图和甘特图比与团队成员交谈更重要；还有一些经理承担了太多的实际开发工作，而不是去解决好团队的需求。

你们组织如何看待经理的角色？他们会因为“催化”了工作受到赞扬吗？你们是雇佣“保姆”还是“管理者”？

模式 6　牵涉性疼痛

项目治愈了外部的病状，却没有根治内部的病因。

*牵涉性疼痛*这个术语通常是指疼痛不是表现在病源位置，而是表现在身体的其他部位。举例来说，脊柱损伤患者并不会觉得脊柱有问题。坐骨神经痛也是如此：病人感觉到腿部疼痛，但是问题却是因为椎间盘脱垂压迫脊椎神经导致。你可以费尽心机治疗病人的腿脚，但那并不能消除疼痛——因为病因在其他的地方。类似地，患有心脏病的病人通常会感觉左臂疼痛，治疗左臂对于挽救病人的生命没有任何帮助。



在项目立项的时候，人们往往会解决显式的问题——最明显的、客户疼痛感最强的问题。但是，如果只盯住牵涉性疼痛，项目交付的产品——一旦交付了——最终就会变成极大的浪费，因为它对于解决真正的需求几乎没有任何的帮助。

考虑下面的例子：忘记密码的银行客户向安全部门申请重新设置密码。在确认新密码生效之前，会牵扯到极其复杂、成本昂贵的用户鉴定流程。在一家英国银行，每年花在重新设定密码的开销都会超过 400 万英镑。于是，一项旨在构建新的软件以简化密码重置流程、降低密码重置成本的项目开始了。

项目开始解决牵涉性疼痛，而不是问题根源（太多人忘记他们的密码）。由于巴洛克式密码设置协议，用户最终得到的密码并没有保存在他们的脑海里面，这家银行收到的“我忘记了密码”的请求远远多于其竞争对手。如果银行解决了真

正的问题，从成本的角度上看，它就能极大减少请求的数量，现有的密码重置流程也就能非常良好地继续运作。

为何只解决牵涉性疼痛而不是病源？观察到的一个普遍原因是不愿意去做调查研究。有时是因为组织的文化，有时是因为项目需要立即着手进行："听着，我非常清楚我所要的。所以，给我生成这些报表就够了。立刻！马上！"在很多组织的内部，分析师需要非常有勇气才能提出问题："如果这些报告在手，你会用它们做什么？你真正想做的是什么？"

有时，我们只想在最亮的灯光下找寻。比如说，我们可能倚重自己所具备的技术，用与我们所熟知的解决方案最契合的方式来看待问题：问 Web 服务设计人员如何解决业务问题，他通常会给出基于 Web 服务的解决方案；问数据库设计人员，你会得到从数据库角度产生的解决方案。不消说，二者都会轻易地忽略其他与自己所偏好的解决方案契合得不够完美的方面。此外，我们可能更喜欢解决最吸引人、能够产生最炫产品的问题。不管哪一种方案，工程师都急于针对明显或者明了的问题找出独创性的解决方案，但他的技术能力却用错了方向，不能带来正确的结果。

是否在处理牵涉性疼痛，一个强烈的信号是临时补丁：允许事情出错，然后解决错误。当目前系统需要做一些修正的时候，这些补丁就会骤然出现。临时补丁用来修正问题的外部情况，而没有去修正主要的系统。临时补丁如同创可贴，很少（甚至从不）能帮助解决问题的根本原因。但是，当一条补丁看上去有效，更多的补丁就会被采用，有时甚至每个补丁上面都是一层又一层的临时补丁。每次在采用临时补丁的时候，这种"创可贴"看上去比外科手术更便宜、更划算。

很多问题的根本原因都是细微的，而且通常与表面的症状一点也不相干。但是，集中精力研究真正的需求，解决真正的问题，总是能事半功倍。

模式 7　明日复明日



照片由 Mark Lisao 所摄（markldxb@gmail.com）

每个人都有时间窗，提醒自己立即采取行动并持之以恒，直至工作完成。逾出时间窗的交付日期不会导致任何紧迫感，因此也就产生不了行动的动力。

如果里维尔在午夜纵马驰过马萨诸塞州的村庄和乡镇，呐喊着："不列颠人要来了！不列颠人要来了！就在明年的某个时刻——我不确定具体何时何地，但是我确定是在明年，不列颠人就要来了！"[①]那么又会发生什么情况呢？

① 或许你不熟悉美国的历史和民间传说，据传美国革命之前，在 1775 年的一个夜晚，保罗·里维尔（Paul Revere）跨马行遍各个村庄和乡镇，警告殖民地的居民不列颠军队就要来了，动员他们准备与不列颠人的莱克星顿之战。亨利·沃兹沃思·朗费罗写了一首诗《骑手保罗》（*Paul Revere's Ride*），马萨诸塞州的每位学童都耳熟能详。诗篇的最后几句是这样的：

"保罗·里维尔跃过了黑夜，
一如他的警报跃过黑幕
传到米德塞斯的每个村庄和农场，
蔑视和无所不惧的呼喊，
黑暗中的一道声音，门上的一次敲门，
一句永远传诵的话语！
从历史的晚风中传来，
经过我们所有的历史，传诵至今，
在黑暗、危急和必要的一个小时
人们被惊醒，倾听，
骏马的蹄声气势如虹，
以及保罗·里维尔的午夜警报。"

对保罗来说，不存在"明天"，他对紧迫性有天生的直觉。

你能预料到他肯定不会获得期望的回应，倒很可能会有人愤怒地吼道：“闭嘴，保罗!”甚至偶尔还会被人投掷夜壶。

紧迫感是实际行动的重要催化剂。如果事情不再紧迫，它在今日待办事宜的列表上就会被排到后面。其他事情的紧迫性就更为显著，今天就应该处理所有那些需要几天才能完成的事情。

每个人都有时间窗，提醒自己立即采取行动以完成工作。对于大多数人而言，这个时间窗的长度是 30 到 90 天。我们可以分析现在的处境，预料出未来 30 到 90 天的工作进展。我们可以计划这段时间的工作，感觉到紧迫度。我们即刻着手，目光牢牢地盯住需要完成的事情。

除此之外，逾出时间窗的日子都是“明日”。“明日”是这样一种状态：意识到自己需要负责完成工作，但却没意识到如果期望成功，自己就必须从现在开始。

大多数项目的周期都长于人们认知紧迫性的时间窗。当组织告诉他们在未来 30 个月之内完成项目的重要性是多么令人难以置信时，人们在内心深处并不能感觉到紧迫性。他们听到了这个消息，他们理解了它的重要性，但在脑海中，有一个声音在告诉他们：“30 个月，那时你也许都见马克思了。”

大规模的项目通过让大多数人保持对时间窗的关注，来避免“明日”的影响。他们把工作转化成在 90 天之内——通常在 30 天内——能够交付的具体任务。你能听到如下的交付建议。

> “在接下来的两周里面，让我们只是针对证券交易员用户创建交易界面的原型吧。”
>
> “让我们写一些代码，使得系统能够接受新的订单，检查订单项是否有库存，然后发送一条处理消息。不用去管订单修改或者取消，抑或其他的事情，只是负责新的订单。我们要能在这个月中旬演示这个功能，觉得怎么样？”

对于项目成员，其好处在于他们在处理这些短期任务的时候，真实的终点线如同就在眼前。在基本正常的项目上，人们努力工作，集中精力于两周之内需要完成的原型任务，就好像他们要在两周之内交付最终的系统。

但是，要小心，关键在于每个时间窗都必须产生真正的交付物。只有进度是远远不够的。诸如“让我们在五月底完成 50%的规范”这样的任务，并没有提供

一个令人满意的结尾。人们会在心里嘀咕：“百分之五十——也就是说五月底不能全部完成了。剩下的工作到底什么时候才能完成？”

警惕“明日”的另一种特殊变体：耗费大量的时间来准备开始。每个人花时间去寻找完美的支持测试的工具；每个人不厌其烦地讨论如何制订库文件，以向开发人员提供最完整的支持。如果这些时间被节省下来，匀到项目的结束阶段来完成工作，它们的价值会更大。

回忆起自己在苹果公司担任项目经理的日子，Sheila Brady 对于“明日”与时间窗有这样的评论：

> “所有的项目，一旦式微到后期，即使时间表上延长一周，也依旧摆脱不了失败的结局。”

就像所有优秀的项目经理一样，Sheila 意识到项目前期阶段的时间并没有像结束阶段的时间一样被仔细地对待，使行动开始的最佳方式就是不等“明天”，今天就开始动手。

模式 8　眼神交流



当任务紧迫而且复杂的时候，组织往往会把项目成员安置在一起工作。

如今，项目分布式开发合作的潮流已经大行其道，也没有任何迹象表明这种做法将会在未来消失。你与我们一样了解这一事实。也许你已经发现自己在寻找理由，向手下的人解释为什么大部分的员工都在这个市区，而在 Kissimmee[①]和 Richmond- upon-Thames[②]却也有几处的偏远办公点。都是因为想充分利用金钱和资源，对吧？

现在，还是痛痛快快地承认吧，如果项目的成败对你性命攸关，你难道会不想把所有的项目成员都安置在一个地方，让他们可以在视线范围之内看到对方？当然，或许远方的团队有一些人身怀特技，其他地方的项目团队没有人能与之媲美。如果这样，你可能会愿意付出分布式办公的代价，否则就不然。关键在于，之所以分散工作应该是因为缺乏人才和技能，而非金钱和资源短缺。而且，工作

① Kissimme：基西米，美国佛罗里达州的一座小城市，位置相对偏远。——译者注

② Richmond-upon-Thames：泰晤士河畔里士满区，是英国大伦敦地区的自治市。——译者注

越紧急，团队成员就越有必要在一起。

当所有全职的、专注的项目成员处在同一个屋檐下工作时，会产生一些很神奇的效果。他们了解彼此的需要和能力，而且随着了解的增多，他们会调整自己的行为方式，以获得最佳的整合效果。比如，这种团队合作的理念与从运作良好的曲棍球队中观察到的现象非常相近。有一种无形的信息交流使得人际之间的合作变得顺畅，而这依赖于人们在地理位置上的接近度。

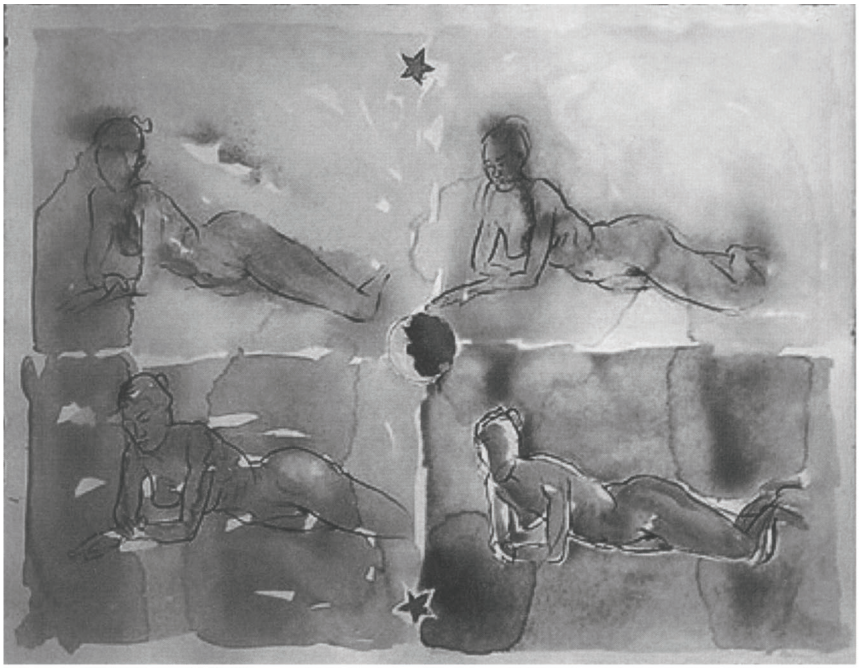
类似地，在开发团队里面有一些关键的信息交流对于紧密的合作也非常必要，其中最重要的是信任的给予和获取。你可以通过电子邮件和电话与远方团队的成员进行交流，把某些事情了解得非常清楚，比方说，对方要求的规范文档，以及他们应许和请求的承诺。如果问你是否相信对方向你表达的信息，你也许会说相信，为什么不呢？但是如果问你："你在多大程度上相信自己所了解到的信息？"则根据对方的工作地点不同，答案也迥然不同。对于一起工作的团队成员，承诺的应许和需求的表达还伴随着双方的肢体语言和过去的经历，这些也促进了关系的不断演化。你清楚对方的真正想法。同样的承诺和需求飞过千山万水的阻隔，在接收到的时候就已经丧失了大量背景信息。

中间横亘着距离，双方很难相互信任。同样，也很难察觉语言上的微妙之处、信心、某些讽刺和挖苦、真实意图、信服程度、无望感和无能为力感、轻重缓急以及话外之音。缺乏了这些潜台词，沟通就变得脆弱。纵然明白了大的图景，得出的结论也依然包含着不确定的因素。缺少了这些沟通，一个项目还能顺利进行吗？当然也行，但它肯定没有在一起工作的效果好。

在第 14 项模式中，我们断言不得不采取分布式开发的团队，即使成员之间会面的机会很稀少，也能从这些会面中受益匪浅。目光接触能提升交流的效果。如果项目足够重要，为了利用其他地方的同类资源而采取分布式开发实在毫无意义。与之相反，如果分散工作方式也能被本国或者全球其他地方的人员接受，那么很明显该项目并不特别重要。在可以做到目光接触的组织内，紧急性和复杂性是用来保证项目团队一地协同工作的王牌。

在忽略一地协同工作优势（这不能容忍）的组织里，分布式团队无坚不摧的神话早已根植管理层的思想深处。任何人，不管身处何方，只要在项目开始的时候可以派遣，就自然而然成为新项目团队的候选成员。在这样的环境下面，团队只是挂着"团队"的名号而已。

模式 9　情绪戒指管理[①]



情绪戒指，Bruce MacEvoy

经理不是基于摆在项目面前的风险、决策和问题来汇报项目状态，而是基于团队的活动、付出和热情。

去听听项目经理如何谈论他们的工作，特别是他们如何交流各自项目的状态。这往往在一定程度上能反映出他们管理项目的方式。

下面的项目状态概要出自一位项目经理，我们假设她叫 Donna。虽然这个例子可能有些极端，但它是众多真实报告的缩影。

> “我很高兴地向大家汇报我们正处于此次发布的特性完成（Feature Complete）阶段。虽然有一些人的进度比较慢，但我依旧为团队取得的成就，以及每个人的努力工作感到骄傲。看到办公室里每个人脸上的笑

① 情绪戒指在 20 世纪 70 年代的美国非常流行。情绪戒指背后的思想非常简单：你把它带在手指上，戒指的颜色反映了你的情绪现状。花几美元，你就能弄明白自己的情绪。

容，我相信团队的士气依旧高涨。我很感激能够拥有表现如此优异的团队成员，而且我知道他们在不久的某天就会解决掉剩下的一些特性。

“接下来是一个悲伤的消息，我很遗憾地告诉大家 Bob Jensen 决定离开公司。Bob 在过去五年里一直是我们 QA 团队的中流砥柱，我们会想念他的。Bob 是继上月 Kathy Enright 后辞职离开的第二位 QA。众所周知，Kathy 也是一位非常有经验的测试人员，所以大家能想象我们的 QA 经理要开始忙得汗流浃背了。

“我，同样有一点担心某些新特性上的测试进度问题。现在来下结论还为时尚早，所以我还只是有一些不安。虽然见到好帮手和好朋友 Kathy 和 Bob 消失在落日的余晖里面我很伤心，但我们的 QA 团队一向以善于挺过艰难时期而著称，而且我知道他们以后会加班加点地工作。随着时间的推移，我们会知道更多的信息。”

请注意 Donna 是如何描述项目当前状态的。

(1) 她更多是在谈高层次的活动、团队成员的认真努力，以及每个人对项目的热情。

(2) 她关注的是当前，以及之前或之后一点点时间，没有在时间、资源和交付物的整体框架内谈论团队的状态。

(3) 当她识别出不符合计划的事情时，她谈的是由这些事引发的个人情感，比方说，“我，同样有一点担心……测试进度问题”。

(4) 她的观察大体上是开放结论式的和未确定式的，比方说，“我知道他们在不久的某天就会解决掉剩下的一些特性”。

(5) 尽管偶尔有一些消沉的话语，这份报告总体上的语气是乐观的。

你也许会问，乐观的、带一点点情绪化的经理，有什么不好？当然，没有什么不好。但是当项目经理的交流方式非常强烈地倾向这一种极端，你往往能发现下面两个问题。首先，这种风格的报告并没有完全达成最基本（所有的项目状态报告都应该达成）的目的：它没有把我们的注意力集中到那些最需要立即采取纠正举措的内容上，以尽量增加项目成功的可能性。它也没有指出现有哪些情况需要我们在接下来的几个星期里面注意、决策和行动。由于 Donna 描述事情的方式，我们获得的只是从几个方面对工作整体的定性评价，根本没有对任何问题进行清

晰的定量分析。

其次，频繁地使用这种交流方式更是有害无益，因为项目经理仅仅关注开放结论式的、目前进行中的行动，常常对他们努力追求的最终结果没有清楚的认识。他们（和他们的团队）只是尽可能快地一步一步低头赶路。但是这些团队极有可能在项目的最后一分钟发现按时交付根本没有指望，或者他们已经偏离航向太远，最后所能交付的东西根本不是他们所承诺的。

我们之前说过 Donna 的交流方式处于一种极端，现在，让我们来听听 Lisa，她则是处于另一种极端的项目经理。

> “上周 4 月 28 号，我们赶上了特性完成(Feature Complete)的日期。本次发布包括 18 个组件，其中有 15 个已经处于‘特性完成’的状态；两个将在本周结束之前完成。最后那个组件——数据仓库接口——还要再迟一些才能完成，我们估计它得到 5 月 20 号才能‘特性完成’。QA 团队正在评估在多大程度上重新设定优先级才能解决这个延误，或者该延误是否会导致我们建议更改交付日期。在我们下次 5 月 10 号的核心小组会议上 QA 团队将给出一个建议方案。不管怎么样，产品管理领导已经同意我们下个月可以在没有那个特性的情况下开始进行公开 beta 测试。
>
> “截至本周，已有特性的自动化回归测试套件已经完成，现在每晚都会运行。测试通过率是在 80%左右，对于发布周期的本阶段，这是典型现象。我们希望测试通过率在四周后，也就是在公开 beta 测试的阶段能达到 90%左右。
>
> “由于在过去的 30 天里面有两位 QA 工程师辞职，新特性测试的开发工作变得迟缓。相应的招聘工作正在进行，但我们必须做好最坏的打算——假定新招聘人员不能及时入职和磨合充分来支援我们的这次发布。QA 正在评估从产品支持团队借几个人帮忙的可能性。在一周内我们应该能知道这次人员流失如何影响新特性的测试覆盖率，但是现在，我们必须假定我们不能达到进度表上的测试开发完成（Test Development Complete）里程碑了。”

Lisa 的项目状态概要与 Donna 的在如下几个方面有所不同：

(1) 她选择了基于最能反映工作进展的交付物的状态来评述；

(2) 她关注于项目的产出、问题以及针对计划提出的修改建议，并且她指出了必要的纠正举措和决策；

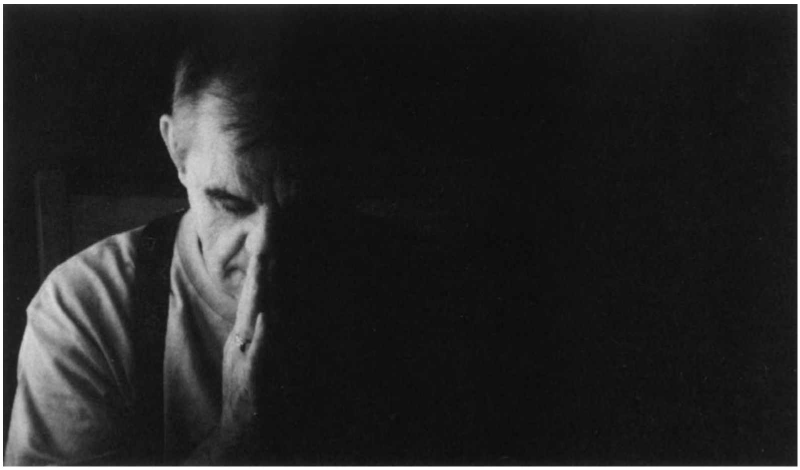
(3) 她的评论是独立的，而非不可切分，而且大部分都是可测的；

(4) 她在客观和主观方面很好地做到了平衡。

很少有经理会完全像 Donna 或者 Lisa 一样交流，大部分都是处于两者之间。不管怎样，Donna 式的经理们还是值得注意：长期关注于付出的努力而非进度本身，有时反映出项目经理没有很好理解领导和管理的区别。

哦，顺便说一句，尤为重要的是从自己身上寻找 Donna 的影子。如果你发现你的个人交流趋于这种方式，就要问自己是否需要这样做，因为正如 Donna，你对团队应该走向何方并不是非常确定，而只知道每个人都在努力工作以求到达彼岸。

模式 10　忠实信徒



个体把某种思想派系作为真理来膜拜，与圣典稍有偏差即被认为是亵渎神灵。

几乎所有流行的软件工程方法学都来源于软件从业者的经验，而不是基础性研究。人们积极地记录下项目里面对他们有效的东西，这些经验从一个小组传递给更广泛的群体。与很多业内领先的流程设计者交流之后，我们得知他们中的大部分人承认：（1）他们的方法来源于一定的领域或者项目大小；（2）他们的方法从来没有被期望如同描述的一样用在所有可能的环境中。

> 我们的 CASE 工具 ProMod 的早期版本里，把很多客户认为 OK 的东西都标识成错误。不管用的是何种方法，我们都是按照书上写的来做——我们是忠实的信徒。无论何时只要书中提出一项建议，抑或展示一个例子，我们都从中抽象出规则，然后作为硬性规定编码进系统里面。如果用户没有完全遵循我们的规则，我们就标识上一个错误。只是随着时间的推移，我们认识到应该把那些东西标识成警告，而不是错误。更进一步，我们认识到应该把所有的消息设置成可选的；换句话说，我们让用户决定是否开启这些设置，并且允许用户违犯规则。
>
> ——PH

虽然大部分的流程书籍对于方法的可用性提出了一定的警告，但忠实的信徒要么忽略这些警告，要么从来不读那些包含警告信息的书页——大部分都作为书的最后一部分内容。如今，拥护 XP 的风潮日盛，那些人中有人甚至都没有读过 Kent Beck 第一本书的倒数第二章。在那一章里面，Kent 很清楚地解释了该方法的不足。

> 我有一个客户因为她的软件工程技能和热情，被她的老板认为是一个主要的成功因素。我们讨论了他们最新版本 2.1 产品里面的UML活动图。让我始料不及的是，她竟然说她拒绝使用公司选择的UML工具，“因为它不支持所有的最新特性，比如*n*维泳道图、中断区域，以及参数集合”。相反，她更倾向于使用Visio图形库，它们在遵循新行为语言的所有建议方面能给予她更大的自由。她声称她的的确确需要所有的特性。她真是一个忠实的信徒。
>
> ——PH

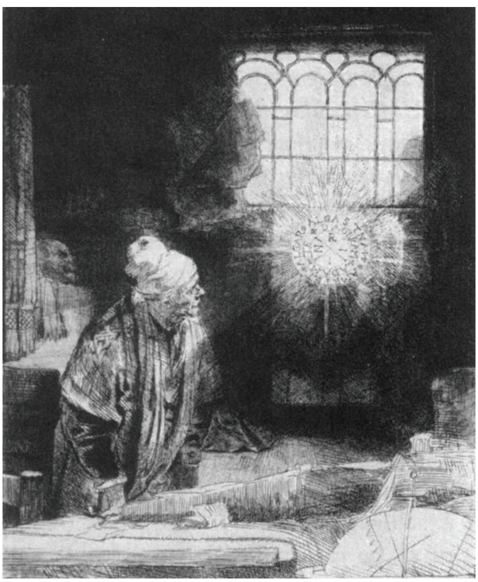
项目上的忠实信徒会让工作止步不前。他们不去专注于内容，反而为方法争执不休。通常，你能在被引入到现场帮助解决方法问题的咨询师中找到忠实的信徒。最终的冲突往往发生在两位主要负责人（内部人员，抑或咨询师）变成两种不同方法忠实信徒的时候。代理人战争爆发！不管他们多优秀，你最好离开他们中的任何一个，以便能继续做你的业务。

> “不同的项目需要不同的方法。”
>
> ——Alistair Cockburn，《敏捷软件开发》

模式 11　出租灵魂

从业者愿意放弃长期练就的技能或者技术。

称职的专业人士有一项令人钦佩的地方，在于能够根据待解决问题的实际情况来裁剪解决方案，而不是把问题往个人或者团队久经检验的技能上生搬硬套。这并不意味着团队成员缺乏应用已知工具或方法的能力。但是，相对于把灵魂出售给任何技术，他们只是出租自己的灵魂。换而言之，一旦出现了好的新思路，他们能够比较各自的优势，非常明智地决定使用最合适的方法。



在歌德的笔下，浮士德签了将灵魂卖给魔鬼的契约

要放弃长期坚持甚至精通的技术并不容易，但灵魂出租者能够忍受暂时的不适。他们知道相对于问题，现有的技术已经绰绰有余，但他们也清楚某项新的技术也许能提供更多的东西。虽然他们并非追赶最新技术潮流的狂热分子，但他们也愿意放弃现在熟稔于心的工作方式，去考虑其他真正的先进技术的优势。他们的态度是着眼未来，而非从现状中寻求安慰。

成为灵魂出租者的好处是当技术潮流退去时，你不会被晾在沙滩上。你可能知道有些自诩为开发人员的人却很多年都不曾尝试学习新的编程语言。这些人痴痴搜寻提到自己所用语言——当年也曾风行一时，但现在基本不再使用的编程语言——的工作职位。悲哀的是，这些开发人员把自己的灵魂出卖给了那种开发语言。

组织要成为灵魂出租者则不是那么容易，但一旦成功，最终带来的好处会让人们认识到付出的艰辛是值得的。诚然，任何组织都不能任意更改自己选用的技术。它需要在开发语言、开发方法、技术基础设施，以及其他方面保持一定程度

的稳定性。但我们这里讨论的是态度。当组织决定对新技术不断研究的时候，它打造了一副金字招牌，吸引着最好的和最聪明的雇员。这相当于在向所有人宣布："这里有一家与时俱进的组织。为我们工作吧，我们不会让你在技术潮流退去的时候搁浅沙滩。"

新的并不一定就是好的。当新技术（编程语言、建模技巧、方法学或软件工具）问世，颇具说服力的宣传通常也随之而来，而且在很多情况下表现为大肆炒作、广告轰炸。有时，新技术被看作是一颗银弹，作为最先进的技术将会产生巨大的进步。在某些情况下，人们臣服于宣传炒作之下，在出售灵魂的过程中变成毫无思想的狂热分子。结果是，他们使用新的技术解决方案去看待所有的问题。灵魂出租者则恰恰相反，他们区分出技术向人们许下的承诺和实际中的效用，并且因为清楚地看到了技术的优势所在，他们能根据新技术的好处相应地进行挑选。

技术正以爆炸性的速度往前发展，今日光彩夺目的创新最后都会沦为明日黄花。灵魂出租者，不管是组织还是个人，对他们使用的技术只是走马观花。在狂热拥抱一项新技术的时候，他们知道这不过是一场盛夏的浪漫。除了使用技术，他们对技术不承担任何义务。他们提出的问题往往是"这项技术适用于什么问题"，而不是"我怎么使用这项技术解决这个问题"。

能够把问题本身跟解决方案区分开来是成为灵魂出租者的第一步。第二步则是要弄明白无论技术多优秀，明天总会有更优秀的出现。不要学浮士德把灵魂出卖给魔鬼那样，贸然订下把灵魂出售给任何技术的契约。

模式 12　系统开发旅鼠[1]周期



虽然组织流程很明显地需要进行定制，但项目团队依旧盲从于未定制的标准。

受 CMMI、SPICE、ISO9000 或者其他流程改进计划的驱动，很多公司纷纷为自己的开发流程制订了内部标准。很自然地，这类流程模型会规定开发团队必须定义的成员角色、团队必须执行的活动，以及团队必须创建的产出物。大多数的流程模型很清楚并非所有的项目都能无差别地对待。因此，它们中有些（比如德国 V－模型和 RUP）也提供了可扩展的定制介绍，允许团队根据项目的实际情况调整角色、活动和结果。

定制流程和（特别是）修剪结果需要勇气。如果你省去一定的步骤或者拒绝创建一些要求的交付物，就等于万一项目失败，你会成为众矢之的。一些批评会很快指出如果你更好地遵循流程，并且创建所有建议的文档，项目可能就成功了。人们出于对指责的恐惧（或者对惩罚的恐惧）而不愿意去定制。最后的结果是，团队出于保险起见而不漏过每个细节，把所有建议的章节和段落都事无巨细地加

① 旅鼠，栖居极地一带的小型啮齿动物，定期集体迁徙，常径直入海溺亡。——编者注

进到需求规范里面，制订质量管理计划（每个里程碑都有相关规定），把任务分解结构（WBS）中的每个工作单元都分派到个人：林林总总，不一而足。

> 当我询问 Philippe Kruchten 如果重新做的时候会有什么改变时，他的大致意思是说如果他能再次开发 RUP，他会让根据具体项目定制流程的工作变得更加容易，并且提供工具支持，以鼓励项目真正去这样做。
>
> ——PH

缺乏勇气并不是放弃定制的唯一原因。通常，其中的缘由更简单。定制流程去匹配项目的实际约束需要做更多的工作。项目经理已经被其他更紧急的项目因素弄得不可开交，哪里还能分身做头脑风暴，为这个项目去建立专门的规范？理由往往像下面这样：公司已经（从项目团队之外）聘用了聪明人来定义流程和交付物。我们为什么要质疑他们的才智？就让我们按部就班地做吧。他们不可能犯那种错。而且再说，没有人给我付钱，让我根据实际约束修改流程。所以，我们别在流程实施上浪费时间了，按照大家做的方式去做就成了。这样的话，我们就能立即开始上马项目了。

如果流程很少照顾到项目的实际需要，照搬流程也许能让项目早点开工，但却无法早点完工。

项目经理不定制流程，就像厨师严格遵循菜谱来烧菜。那样的话，他永远也不会成为一个好的主厨。当然，即便是好的主厨，也是从学徒工开始，从他们的师傅那里学习菜肴烹饪的基本技能，模仿他们师傅的菜谱。但一旦他们掌握了基本技能，停止按照标准菜谱烹饪，他们就能脱颖而出、大放异彩。

模式 13　清空“板凳”



组织变得如此精简，以致于失去任何一个关键人物都会演变成一场灾难。

如果你曾经设定过两个闹钟，或者在汽车仪表盘的储物箱里放一些多余的钱，“只是以防万一”，你知道自己不过是采用了最实用的风险管理手段。天有不测风云，你保护自己的方法就是多预备些重复资源。

如果你来运营一个由专业知识人员组成的项目团队，失去一位关键人物很可能就是你最显而易见的风险。所以很自然地，你也早已悄悄预备下了一个或两个替身，对吗？兴许只是一两个人，他们拥有相关的技能，可以很容易用他们替换项目中任何一位关键的人物。没有？真的吗，这怎么可能？

你没有这些人员储备的原因是他们得花钱（请注意这里，这非常重要）。如果人员储备是免费的，你可能就会大量地储备，但可惜他们并不免费，所以你不会这么做。效率的规定要求你利用尽可能少的人员来完成工作。饥饿法经济可能并不十分有趣，但它的确有效地利用了资源，不是吗？

这种逻辑的问题在于它只考虑了金钱，一点也没有考虑时间。在大多数开发项目中，时间是一项比金钱更稀缺的资源。你的项目，在未来的某个时间点上，

很可能就会发现时间不够。而且一旦这种事情发生了，你和上面的管理层就只能寄希望于舍弃一些小钱来“购买”更多的时间。可是，到了项目开发的那个时候已经太晚了，“购买”时间的机会已经所剩无几了。

留有一些“板凳成员”，在关键人物离开的时候，可能就是一种拿金钱换取时间的方法。如果你深思熟虑地进行人员储备，你也许就能部分地复制某些关键的项目技能。从表面上看，你在团队规模方面会超出最低成本，但是你储备的人员并不会无所事事。他们的能力可能会稍微超出分派的任务所需要的能力，毕竟他们具备在必要时刻像替身一样采取行动的技能。这样的好处在于当你失去了某个人，也许在手边就有一个可以接受的替代人员，并且做好了很快担当该角色的准备。这样，比起事到临头才开始寻找替代人员，你的项目丧失的时间会少很多。

模式 14　面对面



分布式团队通过各地之间大量的面对面交流机会，以建立使远距离团队合作成为可能的熟悉感和可靠感。

> “在计算机学会会议上，总能听到年轻的程序主管声称他们更喜欢一个小型的、由一流人员组成的尖峰团队，而不是由几百个——同时也意味着平庸的——程序员组成的项目团队。我们都这样认为。”

有些人可能会觉得这小段话似曾相识，有些人也许会惊讶地发现，这在 30 多年前就已经首次披露了[①]。今天，虽然头衔由“程序员”变成了“开发人员”，但这样的言论依旧未曾消弭。今天的经理们仍然偏爱那种由明星成员组成的、小型的、一地办公的团队。三十年前如此，四十年前如此，五十年前也是如此，直到今天依然如此——这才是构建软件的最佳方式。

但现如今，大型的分地区合作的开发团队看上去比以往都要更普遍。你也许

① 摘自 Frederick P. Brooks 所著的《人月神话》（*The Mythical Man-Month: Essays on Software Engineering*），第 30 页。影印版由人民邮电出版社 2010 年出版。

经历过团队分布在六个以上地点的项目，不过分布于两三个地点的更为普遍。没错，虽然一个大型分布式团队可以由多个更小的、一地办公的团队组成，但是如果子团队负责的是系统或者产品集成的各个组成部分，则它必须得作为分布式团队来管理。

无论动机如何，分布式开发之所以迅速盛行，可能仅仅是因为协作技术的出现使之不再像以前那么让人望而却步。团队使用即时聊天工具、wiki、视频会议和网络会议以及传统的电话会议和电子邮件，来克服分布式开发带来的重重困难。

那些把分布式团队管理得井井有条的管理者都会精心地为团队成员提供至少偶尔聚在一起的机会。为什么面对面的交流对分布式团队的成功如此重要？如果面对面交流不够充分，在一个地点的团队往往会以高傲的态度对待位于其他地点的团队。“其他地点”通常就意味着“蠢才白痴”。定期的、适度频率的、跨职能的面对面交流能增进友好关系和善意理解，而电话会议和网络会议则会削弱这些。

那么，多大程度上的面对面交流才足够？这一点因团队而异，但下面一般性的指南也许能帮助你对自己的项目做出决定。

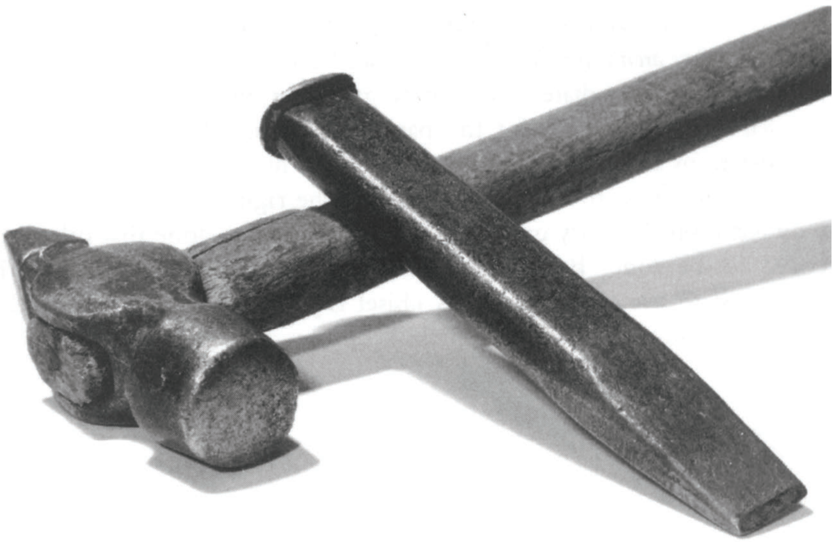
- 在几个地点之间负责协调工作的人需要最频繁的交流。这些人通常都带着程序经理、项目经理或者发布经理的头衔，他们在每次发布周期里面都需要多次面对面的交流，每季度碰面对他们而言远远不够。
- 对于开发人员、QA 工程师和技术文档作者，他们之中的资深人员在每次发布周期里面至少需要与其他地点的同行碰头一次。在这个层次建立双向的可靠感和尊重，有助于这些意见领袖在本地影响初级的团队成员。
- 偶尔允许初级的团队成员前往其他地点。这能帮助他们认识到自己的工作如何适应团队的整体任务，也让他们有机会接触到其他地点资深员工的良师典范，学习其职业生涯规划。同时，在他们凭自己的才华争取晋升的时候，其他地点的同级对他们的高度评价也是非常有帮助的。

在建立远方团队之后，因为削减成本的压力无可避免，组织做出“临时”限制出差的决定，“直到情况有所改善”：这样的分布式开发毫无疑问是百分百失败的。要想分布式开发获得成功，几乎可以肯定的是，必须得增加而不是减少出差预算。

分布式开发天生就是高难度、高风险的活动。有时，接触并留住令人满意的人才可以抵消一部分分布式开发的风险。借用 Tom Wolfe 的名言："我不建议它，你知道的，但它也是可以做的。"[①]面对面交流是一项关键的因素。

① 摘自 Tom Wolfe 所著的 *The Right Stuff*（New York：Farrar, Straus & Giroux，1979 年）。

模式 15　我给了你凿子，可你为什么不是米开朗基罗



经理购买工具，潜意识里希望它们可以赐予团队技能。

通常都是精明的年轻人来销售软件工具，他们对工具在生产率方面的效果，以及给使用者带来的能力上面夸下海口。可以说，大多数客户都看穿了销售人员的大言不惭——什么轻松减肥，什么睡眠时间学习新的语言——但是，有时候 IT 经理不胜其烦，失去了区分现实与幻象的能力。

这些经理承担了交付的压力，而他们却几乎没有人手来做到这一点。自动化工具有时看上去就像是一条救生绳。在某一个绝望的时刻，工具购买者忽视了工具用户必须具备适当技能的观念。

> “工具的成本不仅仅是工具的价格。”
>
> ——Dorothy Graham

我们这里考虑的只是关于表象的问题。工具出现在桌面上，给人一种开发人员能力很强、生产率很高的感觉。但是，工具本身并不能改变任何事情：生产率不会自动提升；报告的错误率仍然居高不下，让人郁闷；而士气也依旧十分低下，令人失望。虽然事与愿违，但生产率瓶颈可以通过开支票购买工具来打破的信念依然顽固。

“你们为什么不是米开朗基罗？”这样的疑问充斥于迫不及待想要生产率立竿见影得到提升的组织、在招聘时看重应聘者的薪水低廉而不是所掌握技能的组织。在米开朗基罗型组织内，几乎总是有各式各样的软件摆放在架子上。

当然，工具是有用的，它们在合适的人手中能带来令人惊讶的生产率提升，并且可以完成那些离开了这些工具就不可能完成的事情。但是正如工具的构造者所告诉你的，拥有恰当的技能去使用工具，这一点才是最关键的。所谓凿子，在米开朗基罗拿起它之前，只是拥有锋利边缘的铁块而已。

模式 16　主面板

Release	Aug 9	Aug 2	Jul 26	Jul 19	Jul 12	Jun 21
Chelsea	Aug 31	Aug 31	Aug 31	Aug 31	Aug 15	Aug 15
Kennington	Oct 20	Oct 20	Oct 20	Oct 20	Oct 20	Oct 20
Kennington Server	Nov 15	Nov 15	Nov 15	Nov 15	Nov 15	Nov 15
Hounslow	Dec 22	Dec 22	TBD	TBD	TBD	TBD
Hounslow for Linux	Feb 14	Feb 14	TBD	TBD	TBD	TBD
Hounslow for Solaris	TBD	TBD	TBD	TBD	TBD	TBD

强团队和弱团队都在使用主面板（Dashboard），但普通团队则不然。

在 Google 里面搜索“主面板”（Dashboard），搜出来的结果很少和汽车相关。在过去的十年里，作为一种可视化和传播项目/业务流程状态的方法，主面板早已家喻户晓。

主面板往往是一份文档或者网页，上面展示了一系列的衡量指标——通常是以图形和数字的形式，全方位描述了项目或流程的状况。有些主面板允许用户“一路点下去”，查看高度概要信息的具体细节。

与众多主面板一样，上面的例子使用了色彩搭配以显示项目或者业务流程各个方面的健康情况。这是一种应用最广泛的方式，它采用了交通灯的三种颜色：绿、黄、红。

通过色彩搭配和简单设计，主面板能提供非常有效和有益的帮助。但是，它们也可能沦为彻头彻尾浪费时间的累赘。无论是哪种情况，这些都与主面板本身无关，而是取决于使用主面板的组织的文化。

最好的团队使用主面板，让大家把精力集中于在正确的时间做出最重要的决定和采取最重要的行动。其背后的信念是：(1)不管何时，总有一些状况会导致项目成功的可能性降低；(2)有时，要识别出这些状况是非常困难的；(3)如果在最重要的问题上采取了及时的纠正措施，项目成功的几率也能提升。良好的主面板能

帮助人们重视需要采取纠正措施的状况。

弱团队则使用主面板以责备其他人或者转移其他人的责备（通过着重强调好消息）[①]。要鉴别是否身处这样的弱团队，不妨找找是否存在着下面的迹象。

- 红色意味着失败。无论这是公开的规则还是潜规则，如果它被团队所信奉，那么团队正在把主面板当枷锁往脖子上套。主面板应该被用来分享项目的状态信号，比如使用颜色强调为了项目成功需要调整的事项。如果把红色看成是不祥之兆，人们很有可能就会把需要公开的状况隐瞒得严严实实。因为害怕责备而避免使用红色，就好比因为烟感器发出巨大的噪音而把它断开一样。
- 橙色其实是不愿承认的红色。弱团队不能正确地处理客观现实。他们想乐观轻松地看待一切事情，而且他们想避免触发警报。这能从他们的主面板一目了然地反映出来。橙色便是戴着瑰色眼镜看到的红色。
- 绿色是站在遥远的地方看项目。经常可以看到，团队不合情理地保持项目一直处于绿色的状态——直到最后一分钟，状态突然变成了红色。在有些团队里面，任何指出项目的某一方面应该标示为黄色或者红色的人，都会被质问有什么证据可以证明该项不是绿色。黄色或者红色隐含着“你搞砸了”的含义。这种文化导致了不可避免的后果，就是所有的项目都一直呈现为绿色，直到失败已经迫在眉睫，才被标成红色，只是此时已经没有任何纠正措施可以采取了。

这些习惯都只是表面现象，反映在深层面中一般都是毁灭性的问题：团队的前进动力并非缘于对成功一腔热情，而是对批评心有余悸。团队成员从他们的领导那里秉承了这一特点。主面板并不能改善领导力，只能揭示真正的领导力。

因此，有效的主面板具备哪些特点呢？下面我们总结了几条。

(1) 主面板不使用海量数据淹没观众。事无巨细地汇报所有的事情必然挫败任何一种项目管理手段。优秀的主面板只提供非常有限的、经过慎重选择的衡量指标。

(2) 主面板可编辑、可选择。从某些角度上看，一个团队的主面板就是该团队的周报。哪些信息应该放在首页上面，哪些应该放在三四层深的地方：决定的过程帮助团队找准关键点。

---

① 参见第 45 项模式。

(3) 主面板上多一些评价，少一些信息。有些主面板看上去就像企业的年度报告，你能看到一页又一页的表格和图形，所有的信息都非常真实而客观，但几乎都没什么用。这样的主面板提供了项目很多的当前信息，但唯独没有指出真正的问题：项目顺利，还是遇上了问题？有效的主面板既提供信息，同时也提供评价。

(4) 主面板在反映现实情况之外，还能帮助预估未来[①]。有些团队把主面板当作记分牌一样来使用，上面尽是已经发生的事实。有效的主面板会同时尝试预估将来会发生，抑或不会发生的事情。比如说有一种主面板就被设计得富有前瞻性，请参阅 *The Balanced Scorecard: Translating Strategy into Action*[②]一书。

(5) 主面板根据时间显示变化趋势。令人惊讶的是，有很多主面板除了给出项目或者流程的当前快照之外就别无其他。除非工作已经完成或者最终结束，否则根本不能在主面板上有效地看到变化的趋势。团队同时也需要知道团队努力的趋势。如果某一项标为黄色，它之前是什么颜色？是因为我们的纠正措施都落到实处，它才变得更好，抑或它正在变得更糟，朝着红色发展？

(6) 主面板在团队需要汇报主观判断时，能够提供一个比较框架。主面板上的评估并非都是绝对客观的；它们反映出——而且应该反映出——团队成员的判断。当它们做到这一点之后，很重要的一点是指出这些判断的含义。你也许会惊讶地发现我们听到了如此之多互相不同且互不兼容的名称定义，比如红色、黄色和绿色。顺便告诉大家，我们更倾向于使用下面这些定义。

- 绿色：计划正按部就班地进展，而且很可能达到预期目标，不需要大的纠正措施。
- 黄色：需要大量的及（或）立即的纠正措施，以满足之前承诺的日期和其他的预期目标。
- 红色：计划已经无法达成了。要么已经错过计划的日期，要么马上就要错过，除非采取激进措施，或许至少还要重新制订计划。

我们曾经在主面板里面使用过绿—黄—红模型，但是这些定义在很多种状态报告里面都使用过。无论它们的样式或者格式如何，有效的主面板都具备下面这一最为重要的特点：它们把团队的注意力集中在那些需要被立即解决的事情上面，以提高项目的成功几率。

---

① 参见第 49 项模式。

② *The Balanced Scorecard: Translating Strategy into Action*，Robert S. Kaplan 与 David P. Norton 合著，由哈佛商学院出版社于 1996 年出版。

模式 17　无休止的集体会议



Evan Lindquist, *Academe:Faculty Meeting*,
©2002 Evan Lindquist-VAGA/NY

允许无休止地争辩，最终肯定无法达成任何一项决定。

在很多项目团队看来，根本就没有最终决定一说。表面上服从决定，随后却继续发出反对的声音。一月份作出的决定推倒重来，一回又一回——二月份过去了，三月份过去了，更多的时间过去了。

这种行为导致最糟糕的后果是把项目最宝贵的资源——时间——白白挥霍掉了。项目领导需要在整个开发阶段做出很多决定。重复不断地对同一个问题做决定，只能说明在它上面花费了太多不值得的时间，却没有在最重要的决定上面投以应有的关注。更有甚者，如果 Marek、Zelda 和 Zac 都把时间花在抱怨他们不认同的决定上面，那他们岂不是浪费了本该推动项目前进的精力？

为什么会出现这种现象？无休止争辩的情况是如此普遍，看起来似乎是起源于组织文化。但是，其实无休止争辩的根源在于团队的领导者。只要领导者容忍一天，不认同决定的团队成员就会争辩一天。这取决于领导者是否认识到该在何时结束这种争辩，勇敢地做出决定。

成熟而专业的组织的一项特点就是拥有有效的决策流程，里面包含了决策制

订和后续事宜处理的规则。美国海军陆战队关于军事理论的简明出版物 *Warfighting*[①]，就包含了一系列被充分理解的决策规则。

> “在指挥官做出决定之前，每位下级都有责任有义务诚实地提出个人的专业意见——即使这些意见可能会和上级的不一致。但是，一旦决定做出，每位下级都应该把它当成自身的决定，毫不迟疑地遵守。”

但是，对于决定，不可避免地会存在一定的分歧。海军陆战队是如何避免无休止的争辩呢？他们都接受这一原则：一旦决定做出，他们就必须遵守。他们认识到服从决定和认同决定二者之间存在区别。但费劲劝说别人去认同自己的看法是没有意义的。如果人们抱有不同的看法，即使某个人做出了决定，也无法改变他们的看法；但遵守决定意味着无论认同与否，每个人都应该采取规定好的行动，而不是把精力放在抗议决定上面。

避免无休止的争辩需要有一个适合具体项目的决策流程。举例来说，产品设计公司 IDEO 意识到需要制订一种流程——在不挫伤创造性的前提下对设计方案做出评选[②]。他们是这么做的：在每个项目的开始阶段，项目团队的各个小组先分开研究问题，并且创建一些设计原型；然后，每个小组向团队的其他成员展示自己的设计原型，每个人都会有一次机会提出他的意见和建议；讨论之后，每个项目成员对自己钟爱的设计进行投票，最终选择投票选出的方案。如果无法达成共识，则由项目经理最终决定应该选择何种方案。这个流程对 IDEO 很有效，因为每个人都知道流程是怎么样的，都知道哪些人为因素可以作为决策流程的输入元素。因此，无论他们认同与否，所有的团队成员都会遵守最终的决定，项目也能继续前进。

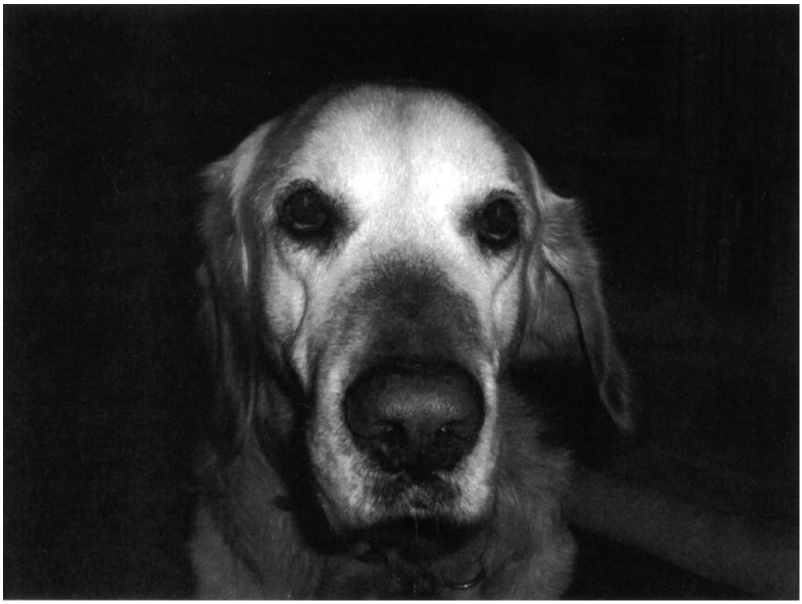
当人们认为只有经过他们同意才能让他们遵守决定的时候，无休止的争辩就产生了。这时，项目经理应该站出来，建立规范让人们遵守决定，而且让人们认识到，一旦决定做出，就必须无条件接受。

---

① *Warfighting*，美国海军陆战队（Washington, D.C.: 1997），第 59 页。

② Tom Kelley，*The Art of Innovation: Lessons in Creativity from IDEO, America's Leading Design Firm*（New York: Doubleday，2001 年）。

模式 18　幼犬和老狗



拥有很多年轻人（20 多岁）的组织比充满老员工的组织更富有生气。

粗一看本模式，你可能会心想："在我们组织里面有为数不少的年轻人。"这是无可厚非的。但如果你读完了括号里的插入语，可能就会想："噢，呃……其实他们也没有*那么*年轻。"

当组织里面活跃着大量的年轻人，工作场所必然充满着努力工作、乐观轻松的气氛，并且能最好地诠释这个词——*趣味*。每个人看上去都很乐意工作。年轻人给组织带来了激情与活力。他们迈入职场的时间还不够长，他们不断学习技能，他们不断学习以使自己成为胜任困难工作的专家。他们充满热情，他们希望工作，而且最好的是，他们希望把工作做好。

所有的这一切让我们这些老员工们不敢停下手中的工作而溜去一边偷偷打个盹。我们必须跟年轻人赛跑，而且通过这样做，我们也变得"年轻"。虽然他们很少能进入组织最高层，但是年轻人却能带动整个组织的节奏。最重要的是，他们在学习方面起了带头作用。他们还年轻，他们在学习。这就是他们应该做的。

而我们这些老员工，大部分知识来自于十年甚至更早之前，跟如今的工作已经不相干。为了能保住自身的价值，老员工们需要赶上年轻人的学习步伐。

把这种组织与“非常老”员工的组织比较一番。“非常老”员工的组织基本上全是由四五十多岁甚至年龄更大的人组成。他们之所以成为这种组织，往往有下面三种原因。

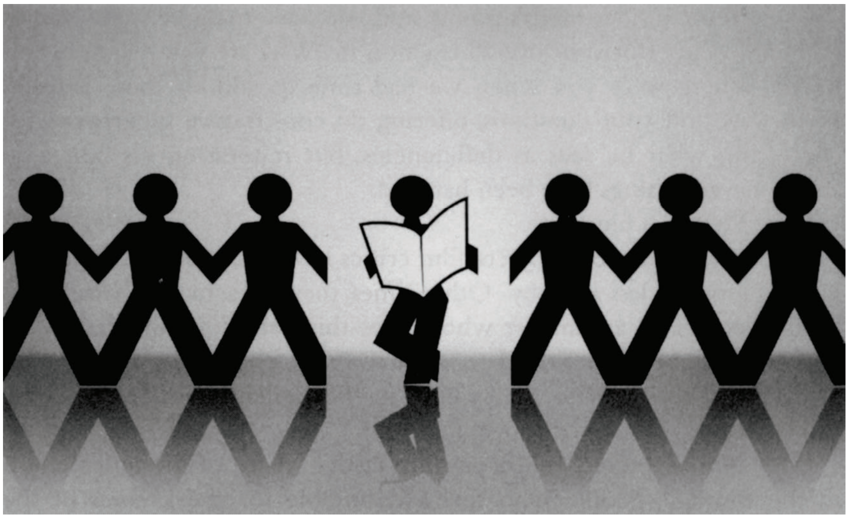
- 组织不再成长，几乎没有机会聘用年轻人。这种情况通常发生在譬如政府机关这些地方，它们往往创建于十年甚至几十年以前，现在只是维护日常的运转。对于有些人，那真是一种清闲的生活，在退休之后还会有优厚的退休金和医疗福利。这一类组织中的管理者常常抱怨很难找到愿意在那工作的年轻人。
- 组织只雇用经验丰富的人。如果长期执行这一政策，组织里面根本就不会出现年轻人。“需要有 15 年 XML 经验，如熟悉 JCL①则更佳”，谁能达到这样的要求？绝对不是刚从校园毕业的新手。
- 组织已全然丧失了冒险精神。它已经无意于采纳那些只有年轻人知道的新技术了，也不会从事充满风险的、可能需要其他新人才的冒险行动。在这种组织里面，身居其位的经理们不希望受到拥有新知的年轻人的挑战。

要想组织停止沦为老员工（或者“非常老”员工）之家，最明显的办法就是雇用年轻人。但这并不是每个人都能做到的：首先，必须要有空缺职位；然后，必须愿意在新员工身上投资。无论如何，招聘一些兼职的大学生总是非常容易的。况且，还有大量的暑期实习生呢。

更何况还有课后工作的高中生呢！

① JCL 是为了取笑古老员工而有意插入的。（JCL 是较早的 IBM 大型机上使用的脚本语言。）

模式 19　影评人



影评人是团队成员或者公司内部的旁观者，他们认为自己给项目带来的价值在于指出问题所在或者将会出现问题的地方，却不把解决问题视为自己的职责。

假设你正处在新系统还有最后几周就要发布上线的紧要关头：集成测试已经全力进行了一段时间；开发人员正在及时修复出现的 bug；发布经理也在一项一项核对预交付事项的检查列表，以确保没有忽略任何东西。这时，在项目就绪情况的评审会议上，一个新的声音冒了出来。通常这位发言人都是在项目初期就加入了进来，但迄今为止几乎没有发表任何意见。我们假设他叫 Herb。

Herb 对事情的现状并不那么满意。他觉得即将交付的产品遗漏了一些关键特性，设计评审也没有完全遵照正确的方式，集成测试应该比当前的程度要严格很多。在讲出发现的所有问题之后，Herb 觉得现在交付系统可能会引发严重的风险。他把这些风险一条条都列在制作精美的幻灯片上，发给了所有相关的人。

仔细来看 Herb 列举出来的问题，你不得不承认并非全部言过其实。但是你总体的反应是："你为什么现在才把这些告诉我们？当我们还有时间处理这些问题的时候，你在哪里？"Herb 拒绝回答你的问题。对于如何纠正他所发现的缺陷，他也提不出任何建设性的意见，只是一味对你们处理事情的方式表示忧虑。

Herb 就是“影评人”。

有些时候，“影评人”在项目上面都有各自具体的工作，他们的评论大概只是出于个人癖好；还有些时候，他们是被那些看重“影评”行为的经理们授予了“影评人”的特权。不管哪种原因，所有的影评人都有一个共同的特征：他们相信即使自己所处的项目失败了，他们也能成功。实际上，他们已经悄悄地与项目团队背道而驰。

并不是所有评论项目的人都是“影评人”，他们之间非常重要的区别在于选择的时机不同。如果对项目的成败负有责任感，人们一旦发现事情做得不对，或者可以做得更好，他们就会毫无顾忌地讲出来。他们会走到他们认为可以发挥作用的人面前，讲出自己的想法。他们尽可能及时地去这样行动，因为他们知道时间总是有限的，纠正措施应该越早采取越好。这些人不是“影评人”，他们是跟你站在一起的“电影制片人”。他们知道如果项目失败，他们也就失败了，因此他们每天都会把问题揽在自己身上，以增加项目成功的把握。你可能同意，也可能反对他们的评论，但你看得出来他们跟你一样，在为同一部“电影”而努力。

继续考衡项目和电影之间的类似点，我们会发现“影评人”往往到“电影”结束，或者快要结束的时候才参与进来，此时已经没有足够的时间来采取纠正措施。其实，并不是他们希望项目失败，更多的是因为他们认为自己的成功与项目的成功是泾渭分明的两回事，并且把更多的时间和精力放在让自己看上去像是敏锐观察到了一些显而易见的事实，以及准确预言了一些无法避免的事实。他们不一定是有意而为之，但只要项目看上去还算正常，他们就不在乎项目的成败问题。

为什么有些项目里面大批地出现“影评人”，而有些项目则很少，甚至于没有？这其中只有一个原因：有些组织的管理文化只是强调正确地做事情，而有些组织则强调不许出现任何差池。当经理们更多地关注于不犯错误，或者错误至少不能被人发现的时候，他们向外界传播了明显的（显式或者隐式）信号：对于组织而言，发现别人犯错与正确地做事情同等重要。组织里面天生就带有“影评人”倾向的人们，会恰如其分地响应这些信号，在他们当前的项目里面扮演“自由影评人”的角色，并观察管理层的反应。如果得到允许甚至嘉许，“影评人”就会成倍出现，责任感则会进一步减少。要记住，做一名“影评人”比做“制片人”（换句话说，做一位可以信赖的领导者或者团队成员）要容易得多。只要组织看重“影评人”，这一类人就一定会出现。

在组织的所有层级中都可能存在着“影评”现象，甚至还可能通过多种方式制度化，其中最常见的例子是“非正式影评人”。这种人在项目上被安排有一个角色，虽然一般仅仅是外围的角色。很多影评人都处于员工支持的角色，他们可以站在那个位子上批评很多项目。特别是在病态管理文化的熏陶下，高级领导者甚至可能授意某个部门都扮演团队建设系统的看家狗。

在项目团队里面，“影评”只是一般的破坏性模式（我们称之为“偏离目标”）的一个例子。注意导致影评人出现的原因：一个项目存在多种成功途径的信念。当然，项目本身也许会成功。但是“影评人”（或者特许“影评人”存在的领导者）却把项目目标替换成了一个相关但却独立存在的目标：准确地指出项目的缺陷。指出缺陷并非坏事——很显然不是。“偏离目标”则极具破坏性，因为人们追求的是不相干的目的，它们只是跟致力于项目成功的努力恰巧有一定关联。对项目成功而言，这些人的努力同样也可能是微不足道的，甚至是适得其反的。

模式 20　单一问责



© Tim Pannell/Corbis

项目的每件任务都清晰地映射到仅仅承担单一职责的个体身上。每个人都十分清楚自己承担的职责，以及自己同事承担的职责。

在一家运作有序的餐馆里面，工作职责被条分缕析地分派给每一位团队成员。调味厨师负责配调味汁，糕点师负责制作糕点，服务领班在门口迎接宾客并引导就位，服务员负责斟酒，女帮厨清洗盘子。如果观察这个工作模式，你会发现每个人只是关注自己负责的任务。服务员在把点餐单别在点餐单板上面后，就把一篮面包送到餐桌上。主厨粗略看一遍菜单，然后把菜肴名称大声通告给每一位成员。海鲜厨师烹饪比目鱼，调味厨师配调味汁，蔬菜厨师准备配菜，装盘厨师把

盘子准备齐全。当菜肴准备妥当，再叫服务员进来，把它们送到餐桌上。每个人都知道自己要做的事情和做事的时机，而且也知道希望同事做什么（这一点也同等重要）。整体气氛是活泼的、紧张的，而且目的明确。当你在项目上发现了这个模式，就能感觉得到空气中弥漫着同样的兴奋感和成就感。

实际上，人们是因为承担职责并且清楚职责内容而兴奋。业务分析人员知道别人对自己的期望，测试人员知道他们与交付物之间的关系，业务用户清楚其他人对他们的期望，开发人员知道他们工作始于何处、止于何处，项目经理知道自己的职责就是带领团队和分派任务……团队中的每位成员对他的任务以及评判任务完成的标准都笃信无疑。

在接受一份单独负责的工作（组件、委派、目标抑或行动事项）时，每个个体就会明白并且承担起他的责任，因为工作被清楚定义了预期的结果。他会想："这就靠我了，每个人在这件事情上都依赖于我——我要为*我的*工作负责。"类似的是，团队的每个人都知道其他人的职责，并且会想："我可以放心地向专门负责此事的同事提出这些要求。"

拥有这种模式的组织如果遇上了未能预知的事情，即使没有一个人对此负有责任，也能游刃有余。人们已经习惯了承担职责，因此在承担尚未有人担负的职责时也毫无所惧。

承担单一职责并不是拒绝寻求帮助，也不是拒绝从同事或者其他相关人员那里获得关键性的投入。关键在于指定的人员仍然对约定好的基本元素负责，他明白并且同意"对于该元素，这就是我的责任……"

这与授予工作头衔，并且假设每个人都会以同样的方式去理解职责是截然不同的。在这种情况下，没人真正地清楚别人对自己的期望。其结果是不确定、担心忧虑、信心渐失、白费精力和枉自蹉跎了太多光阴。

一些项目的运作原则是每一件事的责任都由所有人共同承担。这样的想法表面看起来非常值得赞叹，"我们是一个团队。我们戮力同心，如果有什么需要处理，每个人都要负责确保事情的完成。"但毫无悬念，这种方法很少奏效。想象一下由这样的团队来运营一家餐馆。在烹饪蛋白奶油酥的间隙，厨师会去考虑餐桌预订；服务员会在汤里面多加上一撮盐；餐厅领班会清洗剩余的餐盘，以保证 22 号餐桌有充足的餐盘可用——每个人都需要操心（或干涉）每一件事，却无法把其中任何一件事情做好。

之所以能够进行单一问责，是源于这样一种自信，即人们清楚地认识到了其他人对自己的期望。而且它给你提供了在项目里如何推广这种模式的思路。你需要能够定义基本元素的特征，以便它能被明确地区分，并且使其他人能够获知它是否完成了交付。基本元素也许是软件的某个特定模块，也许是负责对开发人员的设计质量提供反馈，抑或是新产品的用户培训。问题在于基本元素要能够被明确地标识出来，以便每个人都能对它有相同的理解。如果能做到这些，你就可以给每个人安排独一无二的职责内容，从而促进每个人的工作、提高每个人的信心，让每个人从中受益。

插曲 项目秘密



“Subtitles”，Alexei Garev 所做

听上去无关痛痒的词句背后，是并不友善的深层含义。

当他们说……	他们的真正意思是
进度表有些激进	我们有麻烦了
我们将在接下来的几个迭代里面弥补延误	我们还是有麻烦
他真是独当一面	他有麻烦了
行动纲领	充饥画饼
高层次的	脱离实际的
快速组建团队	根本不可能
做特别项目的经理	管理自己的桌子
我们来自管理部门，来这里帮忙	（太直白了，不需要翻译）
工作继续进行	我们一无所知
时间会说明一切的	我们无能，而且我们也承认这一点
这是一次学习经历	我们真的搞砸了
我之前说……	我之前说的都是狗屎
编码完成	没有测试
给你权力了	你要为此事受到问责
唾手可得的成果	连 [谁谁谁] 都不会搞砸的事情

（续）

当他们说……	他们的真正意思是
让我们把它放在一边，继续前进	（跟政客说这句话时所表达的含义一样）
现在，听我的意见	我等级比你高
代码变得不可维护	要是我，设计就完全不一样了
我们还在万米高的空中	它还在我的桌子上，未曾触及
Bradley真是我们的全能手	Bradley真是项目上的白痴
达成一致意见	照我的方法做
最佳实践	是由不在这里工作的人们发明的，所以比我们做的任何事情都要高明无数倍
有效利用核心竞争力	不要挑麻烦事做
线下再讨论	永远不再讨论
你使用了新奇的方法	你真是个白痴
测试被证明是主要的瓶颈	他们总是在找bug
有限发布	不包含功能的发布
让我澄清我要什么	新需求来了
我们在考虑我们的选项	只有一个

模式 21　苏式风格



©2007，Milan Ilnyckyj，sindark.com

交付的产品包含了客户要求的功能，但却不受客户待见，很快就被搁置一边了。

越来越多的人通过网站预订自己的旅程。在使用旅游网站的时候，你输入出发地和目的地、旅客人数等，然后从列出的查询结果里面选择一趟航班。如果你想重新查找一遍以核对公务舱是否存在差别，有些网站就会要求你重新输入刚才已经输入过的全部数据。如果你有足够的耐性，或许能找到满意的航班，但也有可能你就变得气恼，放弃查询，改用其他的同类网站。提供的服务如此不实用，这样的公司通常很快就无以为继，但这不能妨碍其他公司犯下同样的错误。

你可能已经使用过一些苏式风格的产品——它们多多少少实现了功能意图，但却让你觉得笨拙不堪，甚至恼怒不已。你会发现这些产品的可用性根本无从谈起；它们的外观难看，手感一般；它们缺乏一些你认为不可或缺的安全特性；又或者它们蕴涵的文化意味让你觉得不安或者无礼。这些都使产品在非功能性需求（能够吸引使用者兴趣的特性）上面败下阵来。这些特性对于产品最终能否得到大

众接受的重要影响，一点也不亚于功能性需求。

想想 iPod 吧。它的风格正好与苏式风格截然相反。在写本文的时候，iPod 是世界上最流行的便携式音乐播放器，占据了 80%的市场份额。为什么它会如此流行？它不是市场上第一款 MP3 播放器，也不是最便宜的，而且，它的功能跟其他的 MP3 播放器也都一样。它的成功在很大程度上来源于非功能性品质——外观迷人、操作简易、存储容量大、电池寿命长、体积小，还有——我们不得不承认——它简直“酷毙”了。这些都是很容易被项目团队忽视的非功能性需求。

非功能性需求之所以通常会被忽视，一部分是因为历史原因。多年以来，系统分析方法始终关注于功能性需求，使用符号来收集产品的功能和数据。这些方法都假定有人会处理产品服务质量的需求，在“分析人员如何捕获非功能性需求”这一问题上面不是一般地含糊。分析人员非常清楚如何详细地描述用例、描绘实体模型或者活动图，但这些对于非功能性需求，比如文化兼容、观感等却没有任何的帮助。

> 在我大概 35 岁的时候，Tom DeMarco 送给我一把剪刀——我所拥有的第一把专为左撇子人士设计的剪刀。它给了我全新的感受：我可以真正左手使用这把剪刀，我还能真正看到正在裁剪的位置。
>
> ——TRL

要把非功能性需求从系统里面分离出来并不难。很多成熟的模板都可以引导你核查所有重要的服务质量类别。成功的团队会把系统性捕获非功能性需求当作流程中的特殊路径对待，比如构建可用性模型和雇用该领域的专家等。

顾客的抱怨会告诉你产品正是苏式风格，如果产品需要返工和修改的次数异乎寻常地多，也是个明显的信号。你也许缺乏顾客的反馈，比方说 bug 修复和功能增强的请求比期望的要少，这可能意味着你的顾客根本就不用你的产品。出现以上种种情况时，毫无疑问，都已经太晚了——你已经付出了开发成本。

如何才能避免构建苏式风格的产品？首先，保证你的项目计划中明确包含了非功能性需求。这听起来容易，但是要记住大部分系统之所以沦为苏式风格，都仅是因为忽略了非功能性需求。除了持续关注之外，对于能够获取用户好感的非功能性需求，要尽量使用早期项目原型以得到有价值的反馈。

模式 22　自然权力



能力吸引权力。

权力往往会追逐能力，聚集在能力周围。在多功能团队里面（现在还有什么其他的团队类型吗），不同的团队成员在不同领域上的能力参差不齐。做决策的权力应该与能力相配。

知识工作与生产工作极其不同。在生产环境中，工人们追求一个共同的、简单定义的目标（“8 个小时之内生产出最大吨位的爆米花”），完成任务所需的技能对于所有人都是一样的。老板通常是技术精通者，而且对生产线以及生产线工作机制的理解也最深刻。因此，在需要决策的时候，老板很自然是做出决策的那个人。

知识工作则是另一种情形：技能是多方面的，从不同角度看问题得到的结论也不同。如果决策牵涉到一个或多个团队成员的知识领域，那些成员才应该是真正的决策者。如果决策影响到团队之外的其他人，具备相关领域知识的团队成员至少需要参与到决策制订的过程之中。

在常见用法中，英语对“权”（Authority）有诸多不同的解释。其中一种含义是，在特定领域学识渊博的人被认为是权威（Authority），而某人主管某一项工作

则被认为是掌权（In Authority）。权威很可能没有掌权，掌权的人却可能不是权威。良性的模式是——无论身为权威的人是否手握权力，都应该由他们来做出每一个决策。（同时，可能是由掌权的人负责监督决策的实际执行，而不是决策。）

与该模式相反的形式同样显而易见：决策的制订流程遵循组织结构，而不是遵循知识和技巧。身居更高职位的人负责做出大部分决策，有时甚至都不咨询一下那些对问题有更深刻理解的人。举例来说，由国家的政治领袖负责对战术战略做出关键的决策，就是这种情形。另一种情形是组织基于价格因素来做出技术的决策，即费用更昂贵的决策由职位更高的人来制订（他通常高于具备技术能力的权威）。

违背自然权力的流动规律，听上去很像是在滥用职权，仿佛全部都是经理的过错。但在某些情形下，相关领域的专家也应该感到脸红。这种情形略有不同，却也后患无穷。经理（特别是那些不那么受人爱戴的经理）会放任自流，为所欲为，以致最后自食其果。而拥有重要和必要知识的人们只是坐在一边，闷闷不乐地默许（别人）在眼前做出错误的决策。除非被特别要求，否则他们不会提出任何意见。这是对责任的放弃，因为沉默即同意。对于错误的决策，沉默的专家与无知的经理们都要负上同样的责任。一旦这种症状病入膏肓，没有人会太多在意。

模式 23　万籁俱寂的办公室



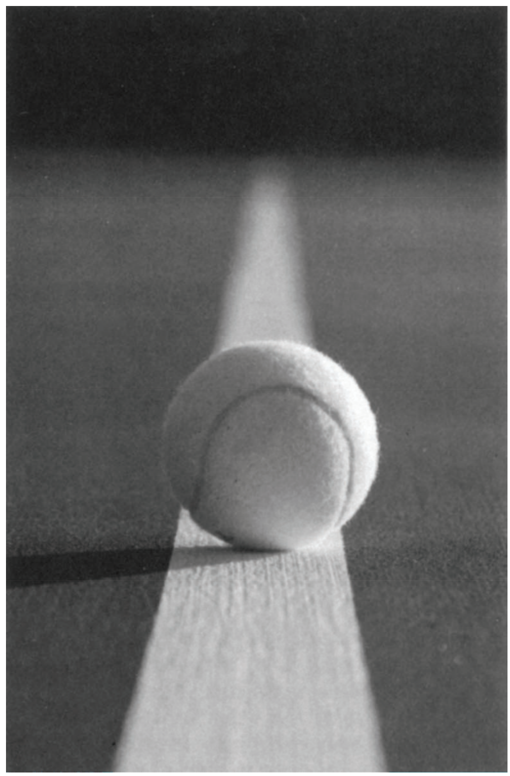
办公室太安静了，凸显出团队已经失去了活力源泉。

只要从大厅走过，你就能对软件开发团队有一个非常深刻的解读。有些团队透出活力：团队成员目标明确、容光愉悦地四处走动。另一种极端情况是团队看上去像是被人遗忘了一样——他们一点激情也没有。人们在办公室里面等着下班回家，等着下一次发放薪水的日子，一边寻找一些有刺激的事情，或者等着退休之日的来到。

模式 24　白线

项目团队借用网球场上的“白线”，来界定需求的范围。

在观看网球比赛的时候，可以看到白线清楚地标记了运动场的边界。任何落在白线之上或者白线以内的球都算为界内，其他则归为界外。大多数的球类运动都会在场地上画上白线，并安排巡边员裁决击球是否落在界内。如果巡边员作出的裁决对球员不利，球员虽然有时也会扬起眉毛表示不满，但他们都会尊重白线。裁决不能模棱两可——球要么越过了白线，要么没有。

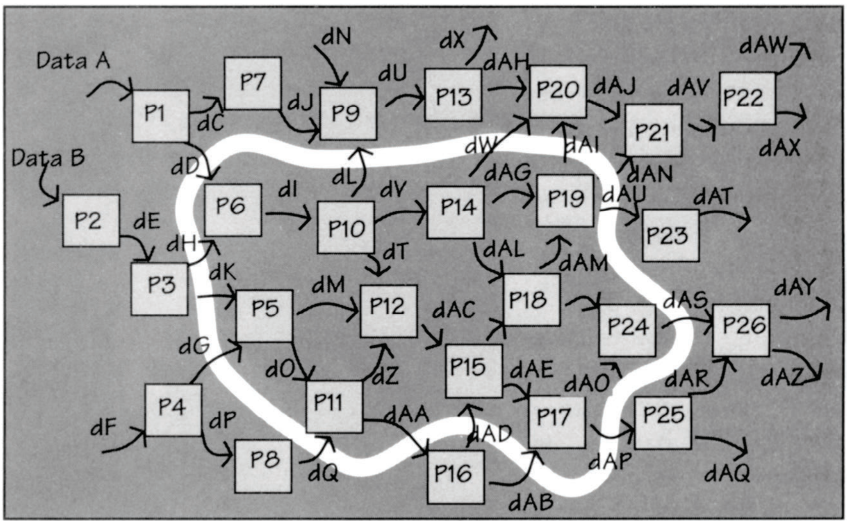


但是很多项目都没有白线。人们试图借助于特性列表或者目标声明来区分哪些需求属于系统范围之内，哪些需求则不属于。然而这两种做法都不够精确，无法成为实用的白线。

特性列表上的任何一个特性固然都需要完成，但却不一定非得完整地纳入需求范围已定的系统里面——它在部分上或者整体上可能破坏了系统的完整性。因此，系统包含某项特性的声明实际上并没有定义系统边界，就像仅凭几张草图演示系统如何生产出界面上的数据，是绝对不可能把特性解释清楚的。

与之相似的是，目标的达成取决于系统内部行为和外部直接活动的联合作用。（比方说，新系统的目标是加快账款处理流程，但该目标仍然依赖于员工往系统输入账款数据的积极性。）声明项目目标往往是值得的，但这样的声明并没有明确界定什么属于新系统范围之内，而什么又超出了新系统的边界。

因此，如何才能清晰地定义系统的范围呢？让我们先看看所谓“系统”或者“业务领域”的本质：它们都是由转换数据的流程组成。这个定义是普适的，适用于任意类型的系统。这些流程通过修改数据的一组属性，从而更新数据的状态，然后再把数据传递给下一流程。下图演示了一系列的流程与数据。



系统内外的流程对数据进行转换，再传递给后面的流程。每一条数据流都有不同的名字，以反映特定的一组属性。通过横贯这些流程，白线把系统包括的功能与外界执行的功能清晰地隔离开来

在不同流程之间传递的数据都是唯一的，不可能存在其他处于相同状态、拥有相同属性的数据流。外部与系统的直接交互视为另外的流程。重要的是，系统和外部之间的接口也只是流程之间流动的数据流。与其他所有的数据流一样，它也是独一无二的。

通过声明（更准确地说，建模）每一个跨越系统边界的数据项集合，你其实是在强调：“边界的这一边生产数据，而边界的另一边则使用数据。”从另一个角度看，你是在通过声明需要修改的系统/业务领域与直接交互的外部世界之间的每个接口来定义项目范围。一旦该工作完成，系统范围就将不再有任何的歧义，你已经借助于接口绘出了白线。

模式 25　沉默即同意



利害相干人无法区分屈服的沉默和同意。

任何一个项目都存在着一个“承诺”系统，它把所有的人团结在一起。开发人员承诺遵循计划表，按照能够被认可的质量标准完成工作；组织承诺偿付薪水和福利，并提供必要的支援（基础设施、援助、硬件、通路等），以保障任务完成。

有一些承诺是显式的，其他的一些则多多少少带点含蓄的味道。比方说，你的公司是否真正地向你承诺了将会向你提供与所有繁忙人士的接触机会，这样你才能从他们那里得到必要的信息以及时完成任务？又抑或那仅仅是隐式的承诺？

当承诺者和被承诺者对于“是否给出了承诺，以及究竟承诺了什么”这样的问题持有不同的解释时，“承诺”系统就破裂了。如果组织内部产生了不满，这些不满经常是集中于那些没有达成一致理解的隐式承诺上面。隐式承诺是光影产生的幻觉，随着观察视角的不同，变幻出不同的形状。经理可能这样抱怨：“他答应我在今年年初交付，结果错过了那个日期；后来给了另一个日期，又错过了；然后又给了另一个日期，现在那个日期还是错过了。”开发人员却可能以截然不同的眼光来看这回事：“我从来没有给过他期限，也绝对没有同意过那个期限。”

承诺被误解的情形通常是：一方表达了需求，然后另一方点头示意明白。前者把这种情形理解成一项承诺："我告诉他必须在 12 月 31 号以前完成，这非常重要。"后者则视为痴人说梦："当然，他希望我在年前完成，但那不可能。"通常，提出要求的人的权力更大一些，而且他会依据法律上的一句格言"沉默即同意"来设定自己的期望。如果对这样的人物没有说出"不"，你就相当于说"是"。

"沉默即同意"式的承诺对每个人都不利。双方不可避免地给工作定义出不同的优先级，最终的结果必然是惨淡收场。理论上，问题看上去是易于解决的：人们必须学会说"不"。但现实世界不是单纯靠理论支撑起来的，它充满了复杂的特殊性。最常见的复杂特殊性是在项目已经过度承诺的情况下，新需求依然以惊人的速度接连不断，隐式承诺像兔子繁殖一样指数级增长。在这样的情形下，假设你已经承诺过度，当经理走到你跟前表示需要在 12 月 31 日之前完成另外一件需求，如果你仅仅耸耸肩，谁又能指责你？

要使隐式承诺更易于管理，一个行之有效的方法是公开声明少量的重要承诺：把它们写下来，然后共享给所有的相关人士。在向所有人公开承诺之前，承诺者和被承诺者必须对承诺的遣词用语都达成一致。只有当显式承诺较少且的确事关重大时，这样的做法才有效。

承诺书没有必要，短短的承诺列表即可——上面列出承诺者的姓名，承诺达到的确切结果和确切日期。在这种情形下，承诺者必然把公开承诺的优先级调得高于自己台面上的其他工作。这个时候，"沉默即同意"的规则可以休矣：只有同意，才是真正的同意。

模式 26　稻草人



团队成员很乐于提供“稻草人”解决方案以获得早期的反馈和认识。

稻草人并不是一个抽象模型，而是一个解决方案。它可能没有完整实现，或者还不正确，但团队仍然有意地把它提供给客户，征求他们的意见。这种做法的初衷可以追溯到世界上最古老的关于“系统”的笑话：

> 客户只有等看到了系统，才知道自己真正想要什么……肯定不是那个系统。

任何一个真正有趣的笑话都蕴涵着真理，这个笑话也不例外。客户之所以弄不清楚自己想要什么，是因为他们对自己能得到的东西一点概念也没有。最好的分析师从来不会试图去分析了客户的所有问题后，再拿出解决方案。他们分析得出一些理解，对解决方案的整体或者部分做出最小的投入，然后把解决方案快速交给客户，寻求客户的反馈。

借用 Steve McMenamin 的话，稻草人模型是一种需求钓饵。你给客户一个激发想法的东西，揭示出他们的好恶。这些模型都是快速完成的，而且因为它们是错误的，所以不必花上太多精力。客户评审解决方案——比如说，“选定区域销售主页”界面——的实物模型、原型或者故事画板[①]。这些东西模拟了未来交付的软件的样子，作为回报，客户引导你得出真正的需求。

最好的分析师不会这样问：“你们想要什么？”他们意识到这种问题通常会令人不快。人们讨厌对着一张白纸创造答案，但乐意于批评纸上已经存在的答案。不妨试试下面这个测试。

你会更喜欢接受哪种任务？

(1) 撰写关于建立新的数据中心的利弊的报告，然后提交给执行官。

(2) 评审关于建立新的数据中心的利弊的报告，然后提交给执行官。

大部分的世人都更愿意接受任务(2)。人们天生就是高效的改进者，而只有极少数会愿意从零开始。

> 在一个新项目启动之初，我参加了一个会议。在会上，执行官说：“我不回答问题。”当时我觉得很诧异。但如果结合背景，他其实是在说：“不要突然问我任何我不曾深入思考过的开放式问题。不要让我难堪。”他太对了。他更愿意找一个可以点评的稻草人来帮助他判断对错。
>
> ——TRL

最好的稻草人模型可能甚至包括一些有意为之的错误。分析师故意弄错模型，以保持客户的警惕性，同时释放出一种信号——针对模型畅所欲言的批评都会被完全无保留地接受。这需要有一位脸皮够厚的分析师，他能够把“做蠢事”作为高级方法，以加快收敛到一个具体的解决方案。这是稻草人艺术的最高境界。

无论人们何时迭代式地寻求解决方案，稻草人建模都是非常有效的。在“稻草人”需求分析之外，想想“稻草人”软件设计。向设计团队提供在你脑海中萌发的第一份设计，可能会带来以下三个回报，它们都是有益的。

- 每次评审之后，特定设计背后的设计策略要全部放弃，使用另一个基于不

① 故事画板（storyboard），拍电影时用来撰写或编排电影、戏剧镜头的画板。——译者注

同设计策略的“稻草人”设计来取代前者。

- 在检查过程中，设计人员对设计进行增量式改进，最终达成一个更受人青睐的设计。
- 奇迹出现了，“稻草人”成为最终选择的设计。（我们从来没有看到这个奇迹的发生，或许某天某地它可能发生——但千万不要指望。）

“稻草人”的哲学是尽早犯错、频繁犯错，这样你将会尽快地得到正确的结果。我们面对的问题和欲寻求的解决方案都太复杂了，不能一下子在任何一个人的脑海中完整成形。勇敢一点，然后问：“这主意怎么样？”——即使你知道自己是赤裸裸的大谬特谬。

很多人自认为已经在使用“稻草人”的策略，但是否曾有人指着你的模型大加嘲笑？那才是“稻草人”。

模式 27　伪造的紧急性



仅仅是为了遏制成本，项目的截止期限被强行安排得非常紧张。

逃避风险可不太好，对吧？如果要问新世纪教给了我们什么，那就是诱人的机会只藏身于风险的浪潮之中。如今，这个理论已被广泛地接受，很多组织都不再去逃避风险。但是不知为何，这并没有让所有的事情都好转起来。

当组织不愿意承担真实风险的时候，他们依旧会夸夸其谈。在大多数情况下，对于那些本可以被准确确定为无意义的工作，他们会断言这些工作有风险，以表明自己无所不知。我们都见识过死亡行军（Death March）项目，那些项目把资源投放在不可能实现的目标上面，最终沦为惨痛的失败。表面上看起来，死亡行军就像是由信心满涨、头脑发热的冒险者发起的疯狂行动。然而，一旦揭开表层，底下的情形却往往迥然不同：如果按照慎重估算得出的时间完成项目，那么对管理层而言，项目的成本就太高了。于是，虽然每一个有判断力的参与人员都说项目需要两年时间，但疯狂的管理层还是把计划时间定为一年。当然，这个项目会

被管理层描述成“极其紧急的”，一个“独一无二的管理机遇”。

要识别是否属于伪造的紧急性，我们需要仔细查看项目可能产生的收益。如果收益千真万确、十分可观，那时限一年的计划安排绝对意味着头脑发热的冒险行为，兴许头脑太过于热了。假如果真能得到如此重要的收益，为什么不多分派一些时间认真去做这件事呢？

更普遍的现象是，所谓的收益其实只是微不足道的，这也解释了为什么组织没有投入全部的精力。明显冒进的计划安排实际上只是制约开销的策略。

伪造的紧急性会引发伪造风险。项目惨淡收场，但这还不是最坏的情况。最坏的情况是组织没有抓住真正的商业机会去从事高价值的项目，而那些项目的风险都是值得去尝试的。

模式 28　时间清除了你的手牌



时间[①]是位拙劣的项目经理。

经理在项目初期的决定对项目的影响最大。比方说，经理通常需要及早——不必在一开始，但是要尽早——决定项目团队的人员配备。当为期 10 个月的项目刚刚开始 2 个月时，增加一个开发人员和一个测试人员也许还有助于准时交付完整的功能。然而，当项目仅仅剩下 2 个月时，再增加开发人员和测试人员恐怕就于事无补了。事实上，这样可能还会给团队带来负作用。从项目刚开始 2 个月到只剩下 2 个月，在这段时间中的某一个时刻，时间彻底错失了添加新人所带来的价值。

时间也会在“下一个发布版本应该包含哪些特性”的问题上作出短视的决定。接下来的情形或许很普遍：客户已经被告知系统下一个完整的发布版本将于 11 月底前交付。然而，对于在截止期限之前完成构建、测试和集成全部系统的任务，

① 作者在本章中把“时间”比拟成了一位项目经理。为了把该“时间”与一般意义上的“时间”区分出来，故本章中所有意为项目经理的“时间”都使用楷体标识。——译者注

团队并没有信心，经理同样也是不能确定。当他把团队的顾虑告诉所有人，得到的答复却是："别扯了，现在才 5 月份。全力以赴吧，这就是我们的要求。"

日子一天天流逝，团队看不到任何值得高兴的事情，发布计划却没有做出任何修改。在 10 月中旬的某一天，公司的火箭科学家[①]宣布："项目团队在 11 月份不可能完成所有的事情。"团队对这则"新闻"致以欢呼，欢呼之中夹带着嘲讽。火箭科学家继续："大家无须惊慌，毕竟我们都是成年人了。明天早上我们首先在 Donner Party 会议室集合，看接下来还能做些什么，好在约定的 11 月份期限之前交付某样东西，以保存一点面子。"

第二天早上，火箭科学家在白板上写下了"11 月 30 日——核心版本"，然后开始会议。他转向集合起来的人群，问道："哪些应该算作核心功能？"

在这个时候，你尽力忍住自己不去放声大笑，因为此时此刻，满屋子的成年人都在假装自己还能掌控局势，而时间早已浪费掉了项目所有的可能性。

"哪些应该算作核心功能？"这个问题的答案很简单："迄今为止所有完成的功能都算，伙计——已经 10 月份了！"基于这个结论，你可以得出等式："完成的功能＝核心功能。"所有已经完成的功能都被列为核心功能：这对版本发布至关重要。我们可以向你证明这一点。

接下来的对话，是你绝对不可能在任何"挽救发布日期"的会议上听到的。

> 经理："哪些应该作为核心版本发布？"
>
> 开发人员："呃，Matilda 已经完成了这个花哨的组件，它能随机改变背景色。这个组件已经完成了编码，经过了测试：它可以发布了。"
>
> 经理："不，这不是核心功能。"

既然该功能已经完成，那它就必须包含在此次发布里面。你不由得啼笑皆非。时间已经完全把项目弄成了一团乱麻。等到每个人都意识到 11 月份之前不可能完整地交付，时间已经彻底把项目带向了以下两种结局。

第一，项目最终交付的功能要少于本应该可以交付的功能。如果人性化的经理和团队在 5 月份就决定放弃"11 月份交付完整功能"的计划，他们还能重新排

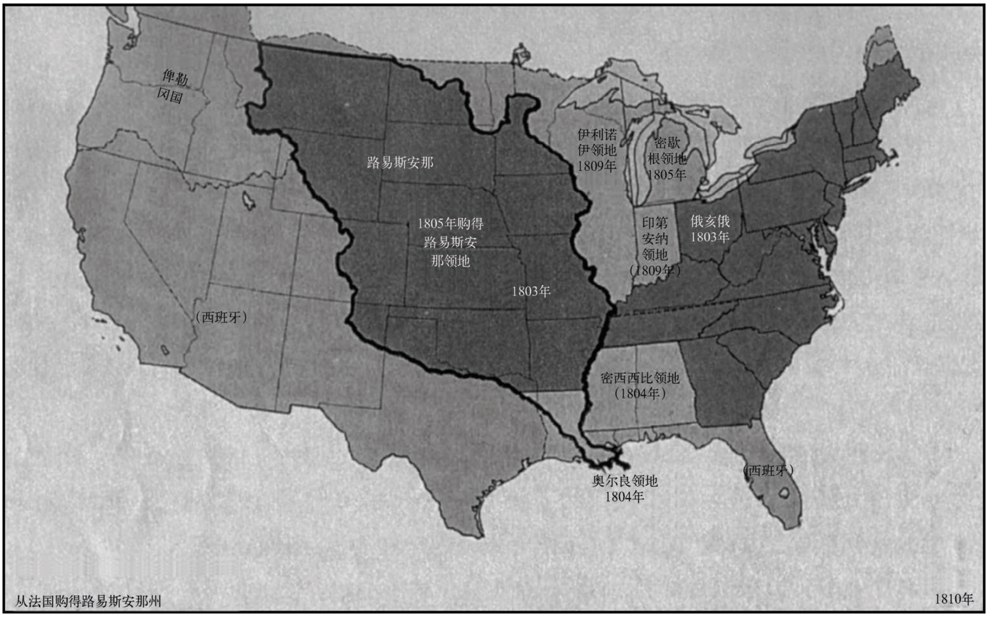
① 火箭科学家（Rocket Scientist），本章指绝顶聪明的人。——译者注

定特性集的优先级，构建并完成特性，按照优先级的顺序依次进行，因为时间允许。他们本可以交付更多的百分百完成的特性，还有一些根本就未开发的特性。从另一方面来说，时间的策略就是生产大量接近于完成、但却尚未完成的特性。然而，接近于完成不等于完成！多么大的浪费。

第二，时间交付面目全非的核心版本。最终交付的不是可以保障 11 月份版本成功发布的最有价值的特性，而是类似于 Matilda 的那种花哨的组件（毫无疑问绝非关键组件）以及其他类似的代码。

所有优秀的项目经理都知道何时需要亮出自己的牌，好让时间无法赢过他们。

模式 29　Lewis 与 Clark



项目团队在前期投入精力，探索领域并发掘潜能。

1803年，美国的国土只包括位于五大湖和大西洋海岸之间的一些州。托马斯·杰弗逊总统向法国购买领土，从而扩展了疆域——此即“路易斯安那购地”交易。关于该项交易，当时的人们只知道“密苏里河流域”的说法。包括杰弗逊总统，没有人确切地知道那意味着什么。在当时，那是一块未知的土地，居住着土著民族，零落地有一些来自法国的捕猎者。为了查明自己代表美国买到了什么，杰弗逊总统委任 Meriwether Lewis 上校与 William Clark 上校带领一支探险队去探索刚刚购买的这片领土，并评估其潜在的贸易和拓居潜能。1804年5月，一支由33人组成的“发现部队”（the Corps of Discovery）从伊利诺伊州出发。这些探险者在1806年9月份回来的时候，带回了沿途绘制的地图和已探地域的信息。Lewis 与 Clark 在他们的日志中详细记载了这些发现（人们如今可以在大部分书店找到这些

日志的不同版本[①])。此时杰弗逊总统拿到了需要的数据，足够做出如何妥善利用这块土地的决定了。而这拉开了美国西进运动的序幕。

一些项目有点像 Lewis 与 Clark 的探索行动：它们分配一些预算对问题域进行探索，以判定可能发生的情形，以及针对该问题域启动项目的切实可行性。

此类探索，如同 1804 年的探险，是纯粹的发现之旅。你无法强求必须发现什么，就像杰弗逊总统无法指示 Lewis 与 Clark 必须去发现什么一样。而且，发现之旅能否得出有用的东西不仅仅关乎运气，还受探索者的技巧影响。另外，就像 Lewis 和 Clark，如果发现之旅找到了某样有用的东西，这次发现或许就会变得非常、非常有用。（如果你是土著民族的一分子，被向西扩张的人们赶出了家园，你也许会不怎么认同最后一句话。）

项目团队中的探索者从抽象的角度来开展探索工作。他们不关心谁在处理什么，或者涉及了哪些机器和个人。相反，他们检查各种条件，看能引发什么点子，产生什么机会，以及该工作未来可能是什么情形。他们寻求着机会和点子，这些机会和点子一旦被证实，将会给探索的发起机构带来最可观的潜在收益。

> 我有一个客户，他的项目请求有一百多个来源。每个请求都包含了不同形式和级别的细节信息，但是没有一个完整描绘了请求者试图达到的结果。问题在于所有这些潜在的项目都包含了未知的领域，在决定采取适当的举措之前，他都不得不再做一些探索工作。但是，这些请求者又总是施压，要求一个确切的成本预估和完成日期。
>
> 我的客户决定对每个项目请求都进行一次短期的探索，以解决这个问题。他使用了一个探索问题的检查列表来分析请求，进而判定请求本身是否已经包含了必要的答案，又抑或必须施以进一步的探索才能量化请求。有时，通过探索结果可以发现：项目的付出与收益证明继续前进是不合适的。
>
> 我的客户现在改变了他的流程，只有短期探索解答了检查列表上所有问题，该请求项目才能真正进入项目列表。
>
> ——SQR

① 想了解 Lewis 与 Clark 的探险故事，请参阅 Stephen Ambrose 的著作 *Undaunted Courage*（New York：Simon and Schuster，1996 年出版）。

“Lewis 与 Clark”型项目把资源慎重地分配到针对业务领域的预先探索上面，以判定项目的可行性。在现实中，有些探索行动的结果显示没有行动能有效地改善情况。这样的发现是一种红利，它阻止了不必要的项目损耗宝贵的资源。其他情况下，探索团队发现确实存在可以把握的机会，从而启动了项目。此类机会带来的收益，就足以补偿偶尔因无效探索而带来的损失了。

模式 30　短铅笔



连续不断的成本削减开始影响到组织完成任务的能力。

的确，成本控制是重要的，甚至是极端重要的。对于组织而言，保持与竞争对手一样低水平的费用开销是至关重要的。即便如此，如果团队成员发出了下面的牢骚，说明某些地方还是大错特错了。

> “我讨厌为这样的公司工作，只有把用短的铅笔交上去，才能更换一支长铅笔。”

即使最热衷于鼓吹削减成本的人，也可能会承认上述措施或许太极端了。很显然，太多的成本削减措施会导致组织缺乏竞争力，最终甚至引起成本的增加。虽然这非常明显，但让我们还是先花一些时间，查看下面一系列成本削减的举措。（这些举措一旦过度，就可能反噬组织自身。）

- 解雇员工，把他们的工作分派给剩余的同事，这最终可能使得那些同事也选择离开，而组织不得不花大价钱招聘新员工，这些新员工又得需要从零开始学习工作。
- 超负荷工作的员工可能会失去热情、请病假、在工作中产生缺陷，以及萌生不满。
- 薪水高的专业人员越来越把他们的时间花在文案工作上，而这些工作原本是由薪酬较低的员工（为了效益，这些人现在都裁掉了）替他们处理的。
- 一线员工可能会因为原来照顾他们的经理们的离职，相应地变得无所适从。

- 人们可能会因为同伴被解雇而怒气冲冲地也离开公司。（而且他们也不会为了公司的需要，去选择合适的时间才离开。）
- 当长期没有任务、计划表一再延期、拙劣任务泛滥的时候，员工的忠诚度、活力、创造性、士气，以及奉献精神都可能会下降。

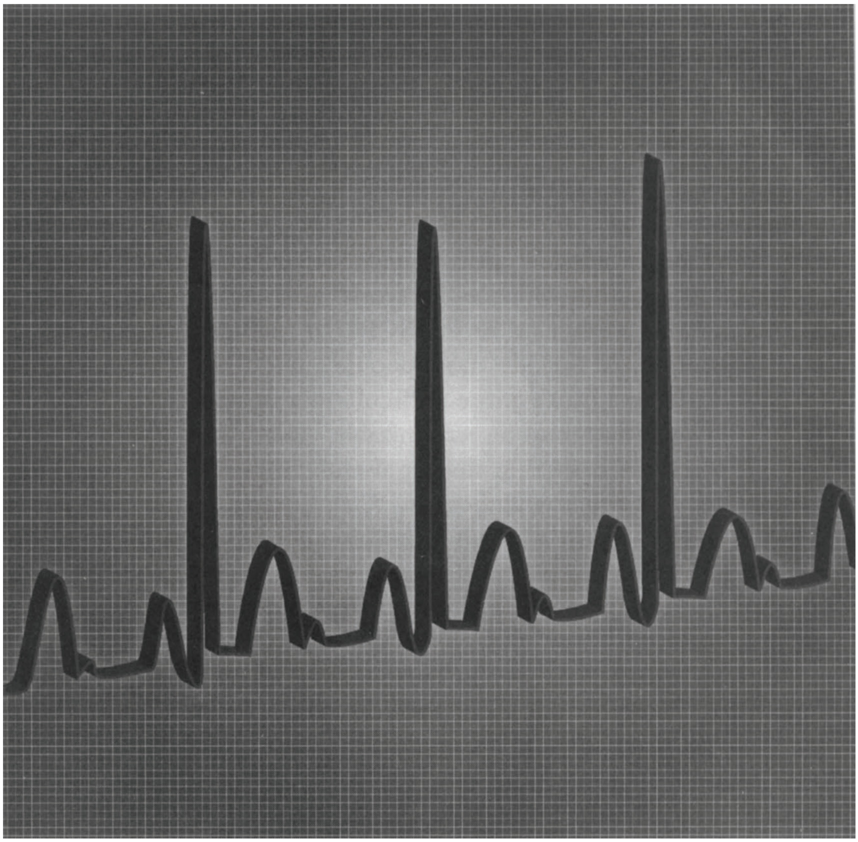
纵览这些令人不寒而栗的事实，再想一想每项效应都会伴随着毫无玩笑可开的、千真万确的成本节省。虽然公司开始体验到全部或者大部分的过度症状，但因为随之而来的成本削减，这些公司或许还是可能见证一个不错的季度。而且，收入尚未受到深重影响，由于成本下降，每一美元背后的收入得到提升——毛利润看上去非常大。但是，这无异于饮鸩止渴。

单一的成本削减计划还可以理解。但是如果你的公司目前将要掀起第二波甚至第三波成本削减的浪潮，或许你该停止思考什么措施能帮助公司削减成本，并且开始考虑自己个人的幸福了。随着长期效应占据主导、收益下滑，为了维持利润，进一步削减成本的争论声可能会又一次甚嚣尘上。也许无论公司的财富是否衰退，你都能保住自己的工作，但那样肯定不会充满乐趣。

“我们需要区分成本削减和组织的贪欲。”

——Ken Orr

模式 31　节奏



团队通过定期交付，建立起工作的节奏。

在从 Chamonix 到 Zermatt 徒步旅行的第三天，我抬头看到一个白雪皑皑的陡坡，这个陡坡一眼看不到边，根本不可能翻越。但是，我的向导无所畏惧，他劝告说："每一步都会让你更接近顶峰。不要只盯着最高点；反之，要集中精力，采取有规律的步伐。找到一个稳健的节奏，然后保持住——你就能攀登到顶峰。"

——SQR

面对艰巨而复杂的任务，拥有节奏的项目团队不是望而却步，而是采取小而有规律的步调，朝着自己的目标发起有规律的冲击。这种明智的团队是这样工作的：首先，他们仰望着“山岭”的顶峰，定下项目的目标；然后，作为一个团队，他们针对一段可预测的时间（通常是一个月），制订交付计划。在那个月里面，团队成员每天都会聚在一起交流各自的进度、想法以及问题，为接下来的一天做好计划。项目目标、每月目标交付，以及每日反馈会议，给工作建立了一种节奏。

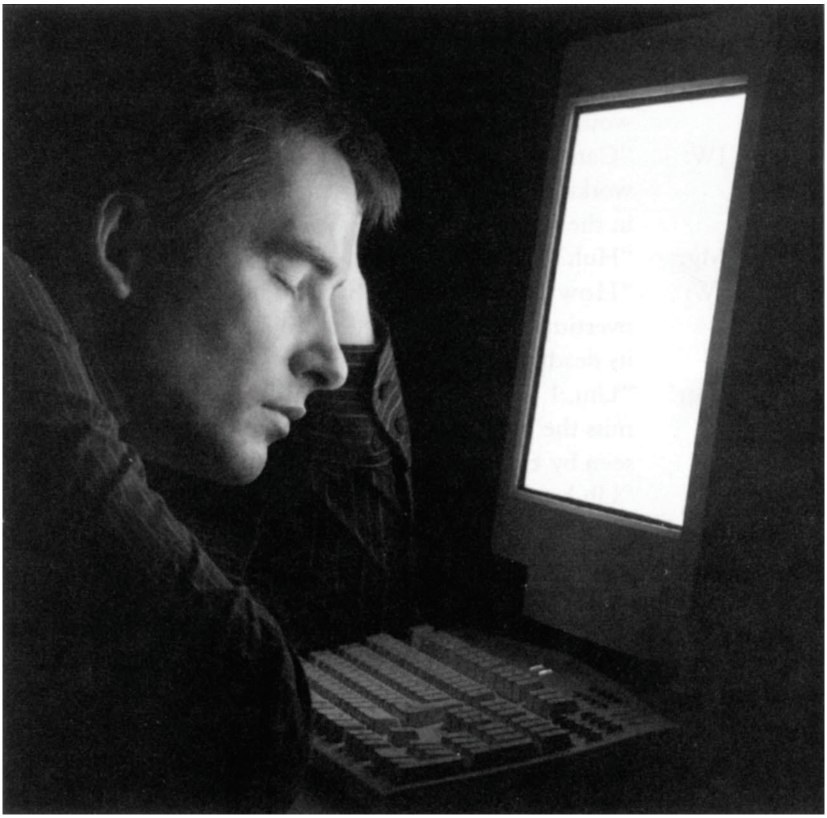
节奏的周期长度并不重要，只要人们能够感觉到节奏的存在即可。因此，一天的周期长度（比如，每日构建），抑或一周，甚至一月（就像 SCRUM 方法里推荐的一样）都是可行的。至于六个月或者更长的时间，则会使大多数人丧失紧迫感（参阅模式 7），回应这种迟滞的节奏，则要困难多了。时间周期必须是可被感知的。

与没有节奏的项目相比，拥有节奏的项目能更频繁、更快速地交付有用的产品。有了节奏，人们就会习惯于按照一个既定的频率交付东西。即使有一些中间交付物不够完美，光凭项目的节奏就能让团队保持精力充沛和热情高涨。没人会期待完美，但是人们都期待交付。不交付，毋宁死。

这种方法之所以有效，是因为每个人都在不断地兑现自己与项目其他成员的约定。按照节奏工作是一种自我强化的行为——团队成员遵照节奏的要求进行生产，与此同时，他们发现越来越容易跟上节奏。保持节奏的压力成为一种积极的推动力。

作为经理，你应该抑制住操纵节奏或者通过加快节奏以提高生产率的欲望。团队建立起自己的节奏，驱动着团队自身进行有规律的交付。无论艰巨任务还是日常事务，如果能够按照持续的节奏去做，它们都会变得易如反掌。想想翻越高峰的目标，采取有规律的步调建立起属于你自己的节奏。

模式 32 加班预兆



经理认为，项目早期的加班表明项目的健康状况非常令人满意。

我们的同事 Jerry Weinberg（以下简称 JW）讲过一个故事，他曾经指导了一个客户——一个初任此职的项目经理（以下简称 Mgr），下面是他们之间的对话。

JW： 对于 Lester，你的集成小组组长，你能介绍一下他的情况吗？你从他身上观察到了什么？

Mgr： 我观察到 Lester 对于按时完成计划非常有信心。

JW： 这是你观察到的？

Mgr： 呃，我观察到的情况是他工作得非常卖力，占用了大量工

作之余的时间。根据这些，我推断出他肯定是认可这个计划；否则，他就不会这么操心了。

JW： 你能不能想一下，假如他不认可计划进度，什么原因会促使他如此超乎寻常地努力工作呢？

Mgr： 什么？

JW： 假如他认定项目会错过截止期限，大量的加班对他有什么好处呢？

Mgr： 嗯，我猜他也许会觉得如果大家都看到他投入了如此多的额外时间，即使最后错过了截止期限，也没有人会去再多加责备。

JW： 啊哈。

Mgr： ……？

经理们，特别是年轻的经理们，都很乐意于看到手下的人把额外的空闲时间投入在项目上。他们把这视为一种信号，表明自己在团队鼓舞和个人激励方面做得很好——每个人正如预期的一样，都是态度坚决地把项目带回家。但是，这也有可能源于人们在工作中产生的无望感：早期的、持续的加班很可能表示项目团队正处于恐惧之中。

如果恐惧文化充斥于组织内部，可能存在多种原因。下面列出了一些常见的引发因素。

- 基于恐惧的管理。有些组织彻底依靠恐惧来维持运转。万一你身处这样的组织，覆巢之下，焉有完卵？离开吧，生命何其短暂。
- 惧怕为了削减成本而强行裁员。如果你所在的公司为了削减成本而开除雇员，或者公司内部风传裁员的流言，你的部下可能会投入额外的时间，希望借此躲避“刀斧”之苦。
- 惧怕个人失败。那些对自己是否有能力完成任务没有信心的团队成员，有时会利用空闲时间加班，而不是请求额外的培训或者指导（这些培训或指导可能让他们收获更多）。
- 惧怕项目失败。如果团队成员对于按照计划成功完成没有信心，他们会在一开始就进行冲刺。这是一件错误的事情，因为他们很难在整个过程中保

持这种冲刺的状态。也就是说，在你最需要他们爆发力的时候，他们可能会疲惫不堪，溃下阵来。

- 确信项目会失败，但惧怕个人承受责备。如果团队成员认为进度计划只是一个天大的玩笑，项目其实从一开始就在走向死亡，那么，有些人就会决定在众目睽睽之下加班加点工作，以确保当惨败最终无法避免的时候，自己不会受到责备。

并不是所有出现加班现象的项目都会陷入困境。在项目交付的最后关头，为了确保达到交付条件和保障产品就绪，这种延长工作时间的现象很平常，也不必因之烦扰。但是如果在项目初期就出现了严重的加班情况，把它视为健康的信号就并不是一种明智的做法。

加班通常看上去像是被工作热情和专业精神所驱动，但真正的驱动力却可能是出于恐惧。早期以及长期的加班预示了并不尽人意的项目结果，包括团队变得精疲力尽、雇员背叛、计划延期，甚至会因为对质量妥协而最终损害产品的完整性。

模式 33　扑克之夜



"玩扑克的人"，Pro Hart 作品

来自组织各个部门的雇员聚集在一起，参加与工作角色并无关联的活动。

如今，对扑克的狂热正席卷整个美国。大约每月一次，七八个人都会聚在其中一位牌友的家中玩扑克。啤酒冷冽，毡桌摆平，筹码堆满，牌已洗好。押注限制德州扑克（Pot Limit Texas Hold 'Em）是最经常玩的扑克游戏。牌友们一般就是工作上的同事，而固定的牌友们往往会邀请客人来加入牌局。

现在，桌子周围兴许就坐着你（主人，同时也是最近刚晋升的产品经理）、两位项目经理、运营部门的副总与她在人事部门工作的朋友、一位开发主管，以及一位由开发主管带来的工程部门的人。与此同时，还有一位来自加利福尼亚州的顾问，他正在帮助在座的一位项目经理处理一些事情。

游戏的名字是德州扑克，今晚唯一确定的赢家是组织。无论人们何时聚在一起，打破阶层和职务的约束，组织都会变得更健康一些。

扑克本身并没有什么特殊的神奇之处，任何一种群体活动都可以起到相同作用，比如社交、慈善，或者社区活动。这种活动可以是公司范围的象棋锦标赛，

或者男女混合垒球联赛。也可以是公司的一群志愿者聚在一起搞的慈善活动，比如“仁爱之家”房屋建造项目或者献血行动。还可以在本地的马拉松赛上，去企业赞助的水站里面担当志愿者。关键是在这种集体情形下，个人在公司中的角色与活动无关。马拉松竞赛选手不关心为他端上水杯的人是经理主管人员，还是收发室办事员。这些人在端递水杯上面都是绝对的胜任。

人们在这样的活动中都是普通的男女，相处时没有职务之分。活动能令人备受鼓舞，它很有趣或者让人觉得满足。这种活动几乎都很圆满，虽然你在扑克游戏中可能输上一些钱，但你不会觉得浪费了自己的时间。在活动中，有很多机会进行海阔天空的闲谈，还有很多机会从其他人那里学习。

那天晚上，你作为主人，发现那位来自人事部门的女士名字叫 Molly，她嫁给了本地广播电台的首席技师，有一对 11 岁的双胞胎。因为 Molly 喜欢扑克游戏，她的丈夫就留在家里照看孩子。自从在普度大学读本科的时候学会了玩扑克，Molly 就一直喜欢这项游戏。Molly 不喜欢啤酒，所以她带来了一瓶非常不错的西拉（Shiraz）红酒，准备和游戏中其他不喝啤酒的人一起分享。

从此以后，当你在停车场看见 Molly，你们会互相招手示意。偶尔你提议再玩，她仍然会试着去说服自己的丈夫照看双胞胎，好让自己可以玩扑克，赚回上次输掉的 20 美元。

终于有一次，你费尽周折，试图雇用一位优秀的应征者到你的团队里面工作，但是人事部门却要求你必须走完所有冗繁的程序之后才能向他提供那个工作机会。你很担心应征者花落他家，于是，你决定给 Molly 打个电话，向她解释这个情况。Molly 告诉你她会看看是怎么回事。她给你回拨了电话，告诉你她因为这事刚与她的老板通了电话。你直接跟她的老板交谈，而那位应征者在当天下午就得到了那个工作机会。

互相熟悉可以使彼此互相信任，也可以使彼此更有耐心。在其他雇员前面，Molly 很容易变得不耐烦。但对于你，她却很难表示出不耐烦的情绪。而且在必要的时候，你也很容易与 Molly 增进相互之间的信任感。

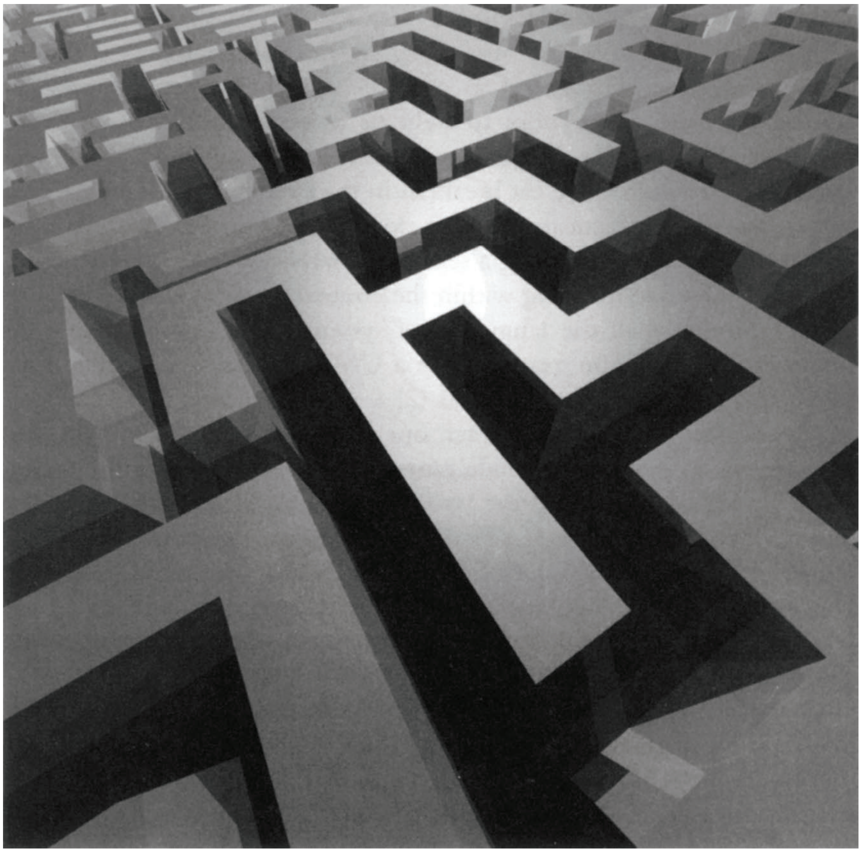
组织内的界线之所以存在，只是为了便于管理和制订决策，往往不是为了加快工作的进度。组织的界线与组织的工作流程通常并不吻合。对于组织中平时不常打交道的人而言，培养他们之间的私人关系对于组织中的重要工作可以起到润滑剂的作用。

很多组织试图通过一系列的团队建设活动人为地“润滑”沟通的渠道。虽然有时也能奏效，但由于个人不能产生志愿参与的感觉，所以大多数情况下的效果并不明显。你签名支持献血行动，其他素昧平生的人也如是支持，这跟得知自己要在周三、周四去异地帮助构建公司形象和精神的感觉完全不同。几乎所有试图强化信任感的组织都会慨叹“蜀道何其难”。

逼迫人们晚上去玩扑克其实也无甚必要。只要创造条件让人们碰面、玩得开心，并且能够达到目的就足够了。无论你做什么，无论这些条件如何形成，无论你如何看待人们和活动，绝对不要去驱迫他们。

> “除了在一起玩耍，我们从来没有在其他事情上面感觉到如此地有活力，如此彻底地展现自我，以及如此地全神贯注。”
>
> ——Charles E. Schaefer 等，*Play Therapy with Adults*
> （Hoboken, N.J.：John Wiley & Sons，2002）

模式 34　错误的质量关卡



项目中的质量保证工作着眼于格式检查，而这些工作根本不能给真正的产品质量带来任何改善。

在抵达里程碑或者结束迭代之前，很多组织都会对项目的产出结果执行规定的质量检查。通常，质量检查分成两部分：一是检查计划制品的完成状态和格式；二是检查制品的内容。第一步是确保预期交付物按照预定的格式完成，第二步则检查各项交付物的内容是否恰当、精确，是否充分达到了项目要求的目标。前一步骤是关乎语法，后一步骤则是关乎语义。

可是，如果缺失了语义，关心交付物的句法纯粹就是舍本逐末。

我们日常交流的语言——法语、英语、阿拉伯语、乌尔都语以及种种——都有它们自己的语法规则。我们使用这些规则帮助发掘和沟通双方的意图。举例来说，英语的语法规定每个句子都必须包含一个动词。“我们今天晚上吃了早餐”是一个句子，也符合前面的句法约束——但是这句话不荒谬吗？要回答这个问题，我们只能进行语义分析以判断该句子在语境下面是否表达了谈话人原本的意图。

与之相似，所有的系统开发语言也有自己的语法。普通的例子是 UML 的用例模型必须拥有一个触发该用例的执行者和一个用例名；设计模型必须拥有每个接口的定义；数据字典的条目必须注明数据范围。但是，这些语法并不能帮助我们检查是否定义了错误的执行者去触发用例，设计接口的定义是否包含了错误细节，以及数据字段的数值范围是否准确这些问题。针对用例的语义检查应该包括询问触发者是否真正地定义准确、该用例的关键触发者是什么，以及在同一流程里面是否还存有其他的触发者。

仅仅是文档通过了语法完备性检查并不能说明它就符合了目标。数据字典中定义了数据字段也不能说明人们就理解了它的内容。如果流程按照“获得输入、做一些事情、产生输出”这种格式来描述，那彻头彻尾就是一种浪费纸张的行径。

> 我们曾经审校了一份功能规范文档，其中正文表格的标题为“术语词汇表”。文档的初衷是规定功能性需求，这样潜在的软件供应商可以对工作进行投标。在阅读文档主体内容的时候，我们发现词汇表包括十项语焉不详的条目——这些条目也与文档其他部分使用的术语不符。但是这份文档满足了质量检查要求，因为它包含了标题为“术语词汇表”的章节。
>
> ——SQR

拥有错误质量关卡的组织关注于交付物的语法和形式，却忽视了内容。关于这种行为，通常有下面三种理由。

- 被指派做 QA 工作的人并不属于项目团队，他们对于仔细研读和评审交付物压根没有兴趣。因此，他们采取了容易的办法，避实就虚，只评审形式。在一个全球性的项目中，某位人士在一份已经发放给所有伙伴的、冻结下一个大版本需求的规范文档上添加了这样的评论：“这几百页的文档包含了

大量的双行行距，因此对我而言完全不具备可阅读性。纠正后再提交过来。”

- 被指派做 QA 工作的人没有接受文档创建以及相关质量检查的培训，又或者他们缺乏领域知识。因此，他们强调文档的标题和编号方式，或者指着作者故意留下的空白行，要求在每个标题下面都应该有一则条目描述。
- 公司的流程模型或者组织结构把质检员工与真正在项目上工作的员工隔离开来，变相地鼓励这种行为。

在很多组织里面（往往是大型组织里面），我发现书面规定了 QA 部门的职责就是检查文档的完备性、一致性以及格式正确性。但是被安排做这项工作的人并不是需求、设计、编程、测试或者系统开发其他任何方面的专家。他们只是“QA 人士”，他们的工作职责就是使用（大量的）规定好的检查列表去检查（大量的）不同类型的文档，然后在不去理解文档含义的情况下对着检查条目打勾。这种公司的流程模型通常都会明确地指出文档内容质量保证的责任在于文档原作者，即那些最初开发制品的人。这些人被认为是他们各自领域的专家。

——PH

如果你使用的是错误的质量关卡，就会发现质检过程传回来的大多数反馈都是关于交付物的形式，而不是交付物的内容含义。该模式把时间浪费在不具备产出性的步骤上面，而且更为严重的是，与内容相关的缺陷却溜进了最后的产品之中。

模式 35　测试之前先测试

“测试可不仅仅是测试（而且应该在测试之前进行）。”

——Dorothy Graham

在传统方式下，测试是等到软件的某些部分完成构建之后再来进行。也就是说，测试人员对已经交付的代码进行测试来判断代码是否工作正确。在非传统的方式下，有些组织把它们的测试活动散布在整个软件生命周期的全过程之中，特别是它们在产品开发的非常早期（远远早于测试对象变得切实可测的阶段——比如软件开发完毕），以及每个迭代的早期就引入了测试。早期测试——也就是说，在测试阶段之前的测试——用来确保项目的计划交付物一旦开发出来后，其正确性能够经受测试。



测试阶段之前测试的理由是它能使得后期测试更为有效，而且自始至终能够有效地减少在可避免错误上花费的修复时间。引入早期测试的组织发现后期可以安心地把精力放在测试产品的运作是否符合预期上面。很多组织无法做到这一点，因为它们对于“工作符合预期”的定义标准没有信心。如果需求本身就没有经过测试，需求又如何能被软件测试人员信任呢？早期测试的目的是给后来阶段的测试提供精准的标准以衡量解决方案。

但是早期测试并非只限定在需求方面，它对任何项目交付物都是适用的。比如说，如果产品设计可以提供某种具体的形式来进行沟通，就能够施以早期测试。项目计划、范围文档等莫不如此。只要所有这些交付物能够提供可测试的形式，它们就都能从早期测试中收益。进一步说，早期测试影响了开发者，使其中间交付物易于被更广泛地理解。

迟到的测试——推迟测试直到产品构建完毕——对项目的成功帮助无几。到

那个阶段，万一错误存在——如果没有早期测试，这些错误几乎肯定会存在——那就太晚了。

在测试阶段之前的测试意味着在最开始的项目讨论时期就引入了质量控制。在采用早期测试的项目中，早期的交付物在下一步骤开始之前都会被测试检查，看是否值得继续。这种早期测试的关键在于尽可能早地揭晓尽可能多的错误概念、误会、冲突、不现实的期望等（在这些观念变得根深蒂固、积重难返之前）。测试阶段之前的测试意味着对影响最深远的交付物进行测试，自然能使你的测试付出收到最佳的回报。

模式 36　苹果酒屋规则

苹果酒屋规则

1. 酒后请不要操作磨床或榨汁机。
2. 请不要在床上抽烟，也不要使用蜡烛。
3. 酒后请不要上屋顶，尤其是在夜间。
4. 上屋顶时请不要带酒瓶。

项目团队成员罔顾或者绕过那些由项目工作无关人士制订的规则。

《苹果酒屋规则》是由 John Irving 写的一本小说，书中讲述了一群在每个收获季节都来到果园采摘苹果酿酒的采摘工人的故事。在采摘苹果的几个星期之中，这些工人住在一间老旧的苹果酒屋里面。酒屋的拥有者 Olive 贴出来一张打印的通告，上面的标题赫然是“苹果酒屋规则”。有一个新来的工人发现大部分的规则都被报以公然的蔑视。在他深究这些违反规则的行径时，一位老工人告诉他：“没人注意这些规则。每年 Olive 都要详细撰写这些规则，但每年都没有人注意过一眼。”[①]

> “规矩是有，但我们从来不遵守。”
>
> ——Linda Prowse

“苹果酒屋规则”的问题在于制订规则的人并不在苹果酒屋里面居住，也没有任何搬进去住的想法，却反而是由她来给住在酒屋里面的人制订规则。Olive 住在大房子里面，根本不知道在炎热的夏夜，屋顶是唯一可以避暑的凉快场所。她根本不理解在屋顶上喝酒已经成为苹果采摘工人生活方式的一部分。既然设定的规定如此脱离实际，她就不要抱怨大家都对这些规定置若罔闻了。在离酒屋很远的

① John Irving，*The Cider House Rules*（New York：William Morrow and Company，1985），p.273。

地方设定规则，并试图强加于别人身上，她其实是在自讨没趣。

一些开发组织也制订了类似的苹果酒屋规则。没有参加项目实际工作的人给那些参加项目实际工作的人制订规则。这样的组织往往存在着一个流程改进部门，或者称为标准组，又或者叫做质量部门，它们的工作就是规定工作的流程或者方法。这些部门兴许还会为项目选择工具，或者为项目的交付物制订标准。这些都是由外行来制订规则指导项目团队应该怎样工作，而他们对于工作甚至都没有一点靠谱的理解。

当遴选流程、方法或者工具变成遴选者唯一的工作职责时，本模式就更为彰显。遴选者不从事项目工作，他只告诉参与项目工作的人应该怎么去做。

外部的规则制订者很少是决定项目工作如何进行的最佳人选。如果对工作不是非常熟悉，他只会泛泛地设定一些规则，而这些规则根本就不得要领。毕竟，他希望做到滴水不漏（其中还包括他自己的小九九）。当出现问题的时候，他的规则必须是可以帮助他避开批评、推卸责任的。而且，规则制订者也不希望其他人认为他的规则在某一方面不够恰当。

成功的项目绝不可能是完全混乱无序的，肯定要有一些规则和定义好的流程。但是，规则制订者眼中的世界和规则遵守者栖息的世界必须得存在耦合的地方。最好的耦合来自于流程和质量专家都是经常从事项目工作的团队成员，或者他们至少是在紧密地关注项目工作的实际情况。一旦满足了这样的条件，专家们就可以作为最佳人选，在整个团队中应用他们的知识，并定义合适的流程。规则制订者需要非常确定所有的规则对于该项目都是正确且恰如其分的，他们的工作重点也就相应发生了转移。

当规则恰当时，项目团队就会遵守它们。这些规则都是非常有用的、合理的，以及看上去有意义的。但当现实和规则格格不入的时候，现实才是王道，那些规则就变成了苹果酒屋规则。

模式 37　说，然后写下来



项目团队在交谈间得出了决定，然后立刻用书面形式记录下来以供交流。

在你主持了会议之后，你应该在会议结束几分钟之后就把会议结果分发下去。乍一看，你会认为这句话的道理再明显不过了。在没看过如此多的团队在这方面做起来五花八门并且饱受折磨的事实之前，我们也认为这是显而易见的。

在需要决策的时候，你希望又好又快地做出决策。时间对于项目开发太短暂了。有时，决策正处在关键路径上面，因为资源遇到了阻碍，或者即将遇到阻碍，就等着做出决策。更经常的是，决策在成为关键因素之前就已经做出，但它们还是需要很迅速地做出。原因很简单——因为有太多的事情需要处理，太多的其他决策需要做出。

对话是快速达成满意决定的最佳方式。井然有序的对话在紧张、顺畅的交流中就把所有人的思想集中在一起。在很短的时间里面，把众多团队成员的经验和智慧汇聚在一起，进而达成集思广益的决议。与缓慢的电子邮件主题讨论不同，高效的对话非常有效，因为它们是同步的，能一直让人投入其中，直到达成决定。

一旦达成决定，就该“换档”了。此时就需要清楚而持久地运用已做出的决

定与所有相关人员进行交流。该项策略并不是全新的。远在 5000 年以前，闪族人（Sumerian）就出于某些需要发明了书写。他们通过对话建立起了长期的贸易和其他谈判，但是他们寻求更有效的方式以保存这些商业和法律事务的结果。于是，他们发明了早期的书写系统，后来发展成为了楔形文字。

即使对于 21 世纪的项目而言，那些曾经在美索不达米亚的市场和庙宇中间行之有效的方法也依然有值得借鉴的地方：书写把对话持久化下来，而记忆不可能做到这一点。用书面方式交流决定，可以为那些没有在场或者忘记细节的人们保存决策制订的对话过程。

紧张对话和清晰记录的好处是如此地显而易见，不得不让人好奇为什么团队不同时采用这两种方法，然后在特定情况下使用二者之中最合适的一种。团队的沟通选择往往会反映出组织的沟通选择，每个组织的文化影响其沟通风格。越大型、越正式的组织往往会更加依赖书面形式，越小型、越快速的组织则往往更加依赖面对面的对话、电话和延续时间更短的书面形式，比如即时信息。这些组织里面的团队已经养成了习惯，它们（深深地）依赖于最契合组织文化的沟通方式。

小型公司的团队在制订决策上面的效率通常非常高：他们的文化推崇使用简短、紧张和专门的会议以解决困难问题和谋划解决方案。但是，他们的文化是如此植根于口头上的沟通，以至于在决策沟通方面他们也试图使用相同的方法。他们是如此满足于口头上的沟通，以至于在有必要“换档”的时候，他们也没有这种意识。

在大公司或者分布式团队里面甚至更容易发现不“换档”的现象。当项目团队的分布地点之间相距遥远，电子邮件通常是重要的沟通方式。团队成员变得如此满足于把自己邮箱里的来信处理完毕，以至于对在电子邮件里面进行提问以得出决定这种做法毫无异议。邮件你来我往，通常也会有越来越多的人被加入到“抄送”一栏里面。那些能够在一两个简短会议上决定的东西争论了好几天也得不到解决。

即使沟通方式与组织文化稍微不合，但如果它们最适合当前的任务，最有效率的团队也会坚持使用那些沟通方式。即使在最正式的公司里，快速和有效率的决策也是通过对话完成的，而即使在最轻捷的创业公司里面，延续性的决策沟通也是记录成书面形式的。

模式 38　项目中贪多求全

组织经常贪多求全就会放慢速度，最终导致净效益降低。但是那种诱惑可能是无法抵抗的……

在这个成本削减和雇员缩编的年代，涌现出了一种舆论（至少在 IT 界是这样的），它认为公司并没有全力开发足够多的软件，因而失去了建立真正战略优势的机会。如果你对这一舆论颇以为然，不妨花几分钟反过来想想：也许 IT 公司正在开发太多的软件。



在 21 世纪，似乎所有事情都应该在昨天完成。如果速度如此重要，一个很明显的折中办法就是通过减轻负担获得速度。不幸的是，如此简单、众所周知的行为与重要的政治“潜”规则大相径庭：推卸任何事物都有可能会激怒某人，或许是某个有权势的人。

假设上级经理 Duane 向你下达了一项工作要求。你知道团队已经处于超负载的状态，但 Duane 的官衔更高，而且他还拥有一口洪亮的声音。如果 Duane 的权势和说话声音足够大，你或许就决定屈服了。

“哎，见鬼去吧！”你内心在叹息，却言不由衷：“当然，Duane，我们会完成您要的功能。”

让我们重新回顾一遍整个过程。组织接受的负载稍微超出了它所能顺利解决的范围，你这样做，为的是能讨得有权势人士的欢心。同样的有限资源现在需要放在更多的工作上面，所以完成那些工作的平均速度会更慢。你牺牲了速度来避免成为 Duane 的靶子。事情变得更糟。Duane 不是你们公司唯一一个有权势的人。事实上，无论是谁，只要他能提出新的系统任务请求，他的权力就比你要高。在

你努力避免指责时，你可能会不由自主地对该项工作说“是”。每次你说“是”，其他所有工作就都受到牵连，变得慢了下来。

接受超出团队处理范围的工作是管理层怯懦的表现。为了避免个人遭受指责，却亲手把团队置于不可能成功的境地之中。最终，团队会饱受超负荷工作之苦，在组织里面受到的尊重度下降，就因为你没有勇气在第一时间说“不”。

怎么做才能逆转这种恶性循环呢？给工作任务排定优先级，只做你最大能力范围之内的事情。把低价值的工作放在一边，先完成高价值的工作。

这可能很难做到。你可以更加快速地交付价值，但却放弃了政治影响力。当你向权力人士说“不”的时候，你的效率得到提升，但你的政治权力却可能下降。背后的规则让人感觉不舒服：你可以让更多的重要工作完成得更快，但只能通过放弃一些潜在的政治权力。

政治并不是组织超负荷的唯一原因，个人也往往会让自己变得超出负荷。他们无法说“不”，他们也听过“少即是多”，但在内心深处，他们认定只有“多才是多”。

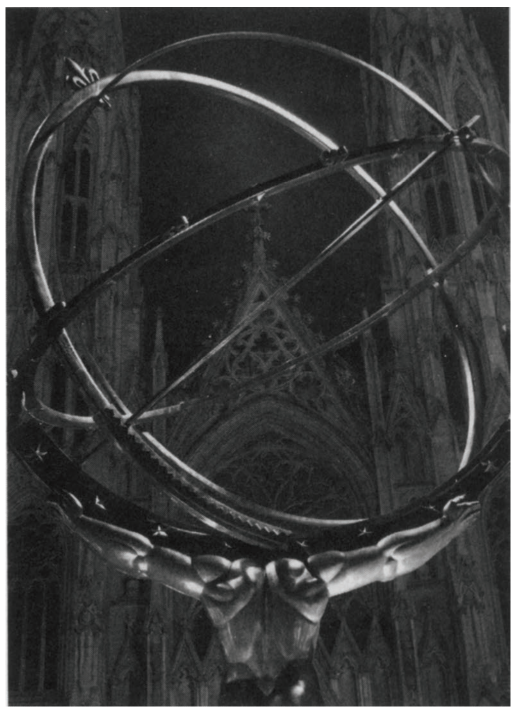
承担超出最大能力范围的工作是变得迟缓的罪魁祸首。你几乎从未见过有人如此坦率地指出数量/速度之间的关系，因为说出来绝对是令人不快的。大家无视这个问题解释了为什么如此多的组织因为试图完成大量的工作而让自己慢到几乎停滞。如果他们暂停下来，从麦麸中分离出麦子，他们就能明白因为如此多的麦麸才导致他们慢下来。

如果你是一位允许本模式（或者通过你自己的行动，或者通过你领导的下属）继续存在的经理，你实际上是把项目置于不必要的巨大风险之中。

模式 39　巨神阿特拉斯

团队领袖（几乎）擅长于一切事情。

你喜欢参观 Erica 的团队。她管理了一个 25 人左右的开发团队，那是公司中最好的团队之一。他们交付令人满意的产品，而且他们也从来不会延期。很多非常优秀的、年轻的新雇员都希望为 Erica 工作，因此当职位空缺时她可以选择最有才能的人。但是空缺机会太少了，因为团队成员都愿意待上很长时间。他们来了，是因为知道自己能学到很多东西。他们留下来，是因为时间一长，Erica 会帮助他们提高技术能力——从最基本的东西到最高级的东西。



不难发现团队的成功在很大程度上取决于 Erica 的领导力。她为她的团队做好了所有的事情。她推动团队做产品设计和发布计划，而且参与了大多数的架构决策。如果一些初级开发人员落在发布周期的后面，Erica 就会站出来，帮助他们按时完成。

Erica 在处理公司方面的事情上也同样富有效率。当应该做绩效考核和薪资调整的时候，Erica 也会亲自动手完成，而不是把这些管理性质的工作交给手下的那些小组组长去做。当她的团队需要与其他地点的团队一起协作的时候，Erica 就是他们团队的代表。实际上，其他地点的大部分开发人员除了 Erica，再也没有碰上 Erica 这边的其他人。

你喜欢参观 Erica 的团队，但是你希望事情能够完全不一样。

Erica 做到了你心目中一个领袖能做到的一切事情，唯独除了一件。作为一个

名副其实的领导兼经理，她没有给团队的其他成员安排任何重要的领导或管理工作。结果自然就是她没有把他们作为领导来培养。

让我们从结构上考虑一下 Erica 这样的团队。她把总共 25 人的团队分成几个更小的小组，包括几个开发小组、一两个测试小组、一些技术文档作者，以及两三个不同领域的专业人士。每个小组都有一名组长。如果这些小组组长选择得当，他们都能成长为未来的 Erica。当然，有一些人不愿意晋升为经理，他们可能希望一直待在某个位子上，让他们可以保持技术上的领先地位。但是有些人会从 Erica 身上，甚至是她上面的经理身上看到自己的影子。为了追随 Erica 的成长历程，他们需要有机会去接手 Erica 的一小部分工作，这样他们就能一点一点地学习她的所有工作。可是，Erica 始终坚持把握着领导和管理工作的每个细节，这样这些潜在的领袖们就无法学习到关键的经验。她喜爱他们，但她在妨碍他们的成长。

我们已经列举了 Erica 团队以及众多类似团队的很多重要优点，现在让我们想想她这种领导风格的另一种结果。最明显的事实已经揭晓无遗：潜在的领袖们都没有被作为领袖培养。但是，另外两种后果可能不会立即显现。

首先，Erica 的团队模型并不能被推广开来。Erica 最初可能只是领导 4 到 6 人的开发团队，那时她可以直接管理到每一个人。通过她的才智、技能和决心，她现在领导着总共 25 人、被分割成 5 个小组的高性能团队。在组织结构图上，Erica 管理着小组组长，小组组长管理着下面的团队成员。但是如果你仔细思考她的领导风格，事情就变得很明朗：虽然她任命了 5 个小组组长，但实际上仍然由她直接管理着所有的 25 人。

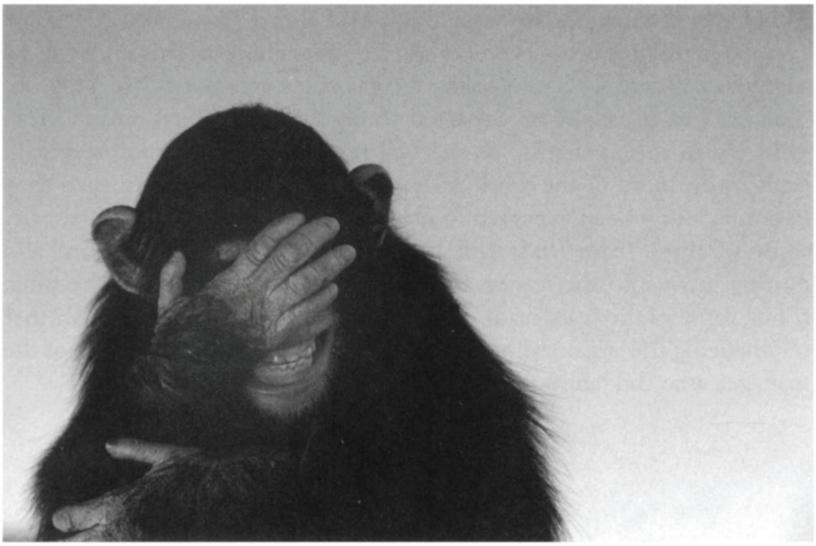
一旦新的挑战出现，要求构建一百人的团队，Erica 的老板能寄希望于她，让她管理这个新团队吗？几乎可以肯定不会。拥有超出常人天分的第一线领导，比如 Erica，可以在 25 人的团队中获得成功，但她的方法在一百人的团队上面无法奏效。尽管这样的领导风格在小型团队中有效，但局限于此的经理们却无法领导大型团队。

其次，如果 Erica 突然离开，怎么办？谁能出马接管整个团队，并使它保持如此优异的表现？如果你是 Erica 的经理，你现在就要发愁死了。你有一个非常现实的选择，可惜存在本质上的风险：你要么从外界招聘，要么把其他经理调过来。不管哪种方法，新的领导都有一段艰难的路要走。

Erica 的团队已经完全依赖于她了。新领导要想快速获得认同和接受将会是一

个很巨大的挑战。更进一步，你不可能期待新领导完美地模仿 Erica 的风格，也就是说他或者她不得不要求小组组长们从事他们此前没有做过的领导和管理工作。如果新领导非常幸运，一些小组组长还能不负众望，快速地成长为他们一直就想成为的领导。如果不是这么幸运，就需要替换一些小组组长，进而伤害了组织，就像组织因为失去那位事无巨细一人搞定的经理而受的伤害一样。

模式 40　所有人都穿着衣服是有原因的



© Tim Davis/CORBIS

完全公开的政策让进度慢慢停下来、停下来……最终完全停下来。

自诩为“公开”的组织通常是非常乐意成为这种类型的。他们自吹自擂，“我们是一个公开的组织”，希望给其他人或组织留下深刻的印象。但是，过分的公开也有弱点，正如智能研究的先行者 Herb Simon 很好地描述过：“信息冗余导致注意力涣散。”[①]当吸引注意力的信息量太多时，我们会处在超负荷的状态之中。即使信息再多，又有何用？

> 我曾经做过一家小型 IT 公司的顾问，所有的雇员都被邀请参加所有的会议。更糟的是，这种会议经常召开。你也许会想知道为什么。一位经理解释了背后的原因：“你必须理解这家组织，Tom，所有人都认为他们必须要知道所有的事情，才能开展工作。”
>
> ——TDM

① H.A.Simon，“Desiging Organizations for an Information-Rich World”, *The Economics of Communication and Information*, D.M.Lamberton 等，（Cheltenham, U.K.：Edward Elgar，1997），pp.187-203。

如果在你的组织里面，所有人都需要知道所有的事情才能完成工作，那你可就前景不妙了。

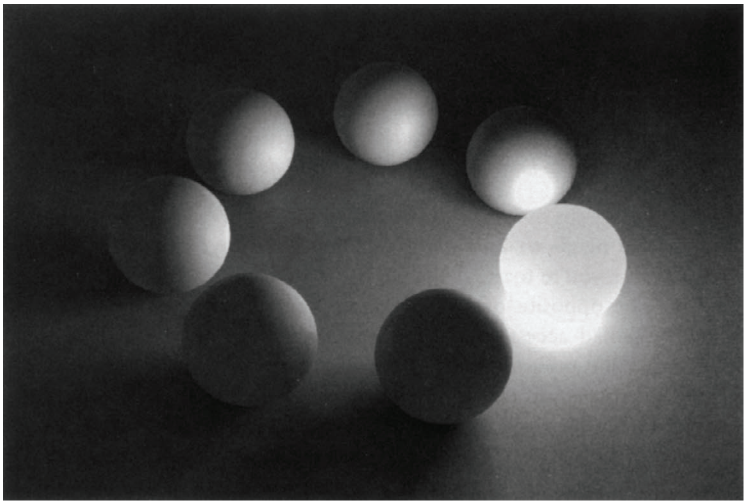
与完全公开型组织相对立的是这样一种组织，在这里你只能接触到需要你知道的信息。这可能是典型的安全性要求很高的环境，比如安全战略或者武器研发。这样的组织由于严苛的访问控制会存在一些不便之处，但它们还是可以运转良好，它们甚至可能比那些完全公开型组织运转得还要良好。

很显然，处于这两种极端情形中间的组织恰好能符合你的要求。个人可以去了解周围项目的一些特殊之处，是很不错的一件事。这种公开性意味着组织认为自己应该促进个人的成长。但是，公开性不应该被最大化。

太多的信息可能是一件坏事，这一点无论如何强调都不过分。但更大的问题在于首先理解为什么我们让自己承担如此之多。一个原因是别人提供的信息可能包含了浮士德式的条件：如果你收到了信息，又不表示不满，你实际上是同意了全部的信息。（参阅第 25 项模式。）

至于我们给自己加上信息的重枷，一个更常见的原因是信息焦虑——害怕自己不知道别人知道的东西。如果出于此种原因而让步，那么你就会像第一次参加婚礼自助餐的孩子——你盛满所有看上去不错的东西，害怕错过任何一种口感体验。知道多少信息适合自己的信息餐碟，不仅仅是一项好的策略，而且是成长的一部分。

模式 41　同事预审



组织让将来与应聘人共事的员工也参与到招聘过程中来。

在如今大部分的组织里面，是否给申请技术职位的人提供工作机会——这个最终决定权属于管理部门。经理们雇人，经理们裁人：一切都天经地义。然而在某些组织里面，这些技术人员能否得到工作机会却是取决于——至少部分取决于——他们将来的同事。这种同事预审的最终结果只有一种：当经理们让技术职员拥有发言权的时候，每一个人——应聘人、职员和经理——都会和盘托出自己的想法。

招聘流程的前期阶段基本上没有变化：通常由第一线经理对应聘人的资格进行最初的筛选。经理也许会要技术方面的高级职员先审阅简历，把所有的应聘人归为“进入下一步”和“直接淘汰”两类。经理也许会给那些在简历上看起来很好的应聘人拨打电话聊一聊，做一个初步审查，决定邀请哪些人过来现场参加面试。获邀的应聘人会被告知他将和团队在一起呆上 3 到 6 个小时。一旦众多的同事参与到招聘的过程中，面试的当天往往非常地漫长。

在抵达公司并与经理寒暄过后，应聘人就随同各位团队成员开始一系列的面试。每轮面试的会谈时间从 30 分钟到 90 分钟不等。

虽然每位参加同事预审的面试官试图从应聘人身上找到相同的信息，但每个人都会使用不同的方式调查。很显然，每个人都希望对应聘人的知识、技术和能力做出评估。所以，针对招聘职位的类型不同，从团队的角度出发也许会要求应聘人编写一些代码，或者创建一组测试。但是，每个面试官都会再以个人的角度去评估应聘人，他会问自己："我能和这个人一起工作吗？他能融入我们这个团队吗？他会让我们团队变得更强，还是更弱？"

此外，每个面试官将以自己在团队之中担当的角色去评价应聘人。举个例子，如果应聘人是一名开发人员，相较于团队里的开发人员，身为测试人员的面试官就会提出不一样的问题，以挖掘应聘人的个性特征。

每个面试官也会根据自己的以往经验去评价应聘人，问自己这样一些问题："这个人展现出的技能与解决问题的方式，是我向来最为看重的吗？这个人在多大程度上让我回想起曾经合作愉快的同事，又在多大程度上让我回想起无法顺利合作的同事？这个人是像他表现出来的那般优异，还是在假装如此？"面试官背景的多样性，使得团队可以从多个视角和价值取向来甄别将来的同事。

在把应聘人交给下一轮的面试官之后，这一轮的面试官向经理——或者面对面地交谈，或者通过邮件，又或者拨打电话——就他对应聘人的印象和意见发表个人的观点。每位职员都需要针对"假如由我来裁决是否雇用这个人"投下自己的一票。

如果面试一切进展顺利，应聘人最终会回到经理这里，而此时经理也已经知晓了所有面试官对该应聘人的看法。经理现在要从他自己的角度出发来面试应聘人，然后告诉应聘人是否得到了工作机会。即使其他的面试官都表示认可这名应聘人，经理也可能会出于某些原因而不聘用他。但是，如果团队的大部分成员都对该应聘人倒竖大拇指，那肯定是没有理由再雇用此人了。

当团队在招聘过程中能真正拥有话语权的时候，就出现了多赢的局面。

- 当前的团队成员是赢，因为新员工加入团队的时候，大部分的团队成员都已经跟他见过面——实际上已经接受了他。不被职员接受的那些人是绝对不可能跨过那道门槛的。
- 应聘人也是赢，因为他处在了更有利的位置来决定是否要加入这支团队。他可以接触到未来的同事，而不仅仅是老板。他可以询问工作的真实情况，同时也可以感受公司的文化。

- 经理是赢，因为他可以借鉴整支团队在技术方面对应聘人做出的评价，而不是仅仅凭借自己的揣测。他也知道团队在一定程度上已经接受了这个新员工，这对他个人的成功也非常重要。
- 最终，团队作为一个整体也是赢，因为团队成员在参与同事预审的过程中，可以向其他人学习。在阅读其他人对应聘人的评价时，团队成员能发现其他人使用的问题和标准，而他们可以在以后的面试中把这些派上用场。经理也能对自己团队成员的思维方式有更深的了解。

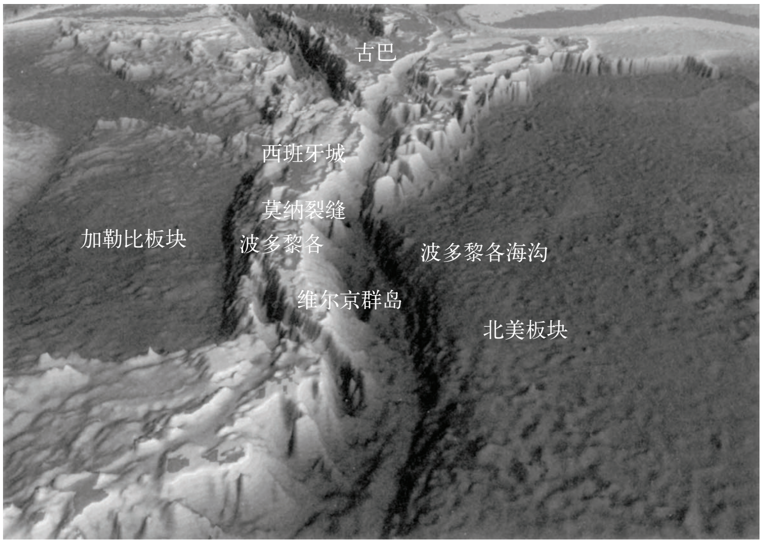
从经理的角度上讲，同事预审在雇用团队组长的问题上面同样有效。为什么不邀请团队里的一些成员去面试未来可能成为他们老大的人呢？

不管应聘人是应聘开发人员、测试人员，抑或是经理，寻找到一位真正合适的人选从来都不简单，但往往又很重要。这是需要团队协作的项目。

> “团队中有人击出全垒打，正是对管理的嘉许。”
>
> ——卡西・史丹格[①]

① Casey Stengel（1890—1975），在 1912 年到 1965 年间先后是美国大联盟棒球运动员和扬基队教练，并于 1966 年入选棒球名人堂。——译者注

模式 42　浮潜与水肺潜水



不同形式的分析活动贯穿项目的整个生命周期：自上而下、自下而上以及先中间后两边。[①]

浮潜潜水员游弋于海水表层，看鱼戏浅滩，望影掠深海。水肺潜水员可以越过海水表层潜到更深的地方，在一定的区域内研究那些影子，试图发现鱼类、沉船残骸以及珊瑚。在相同的时间内，浮潜潜水员可以游历更宽阔的水域，而水肺潜水员则能潜游得更深入。成功的项目团队在项目的整个过程中会把浮潜和水肺潜水这两种方式结合起来使用，在特定的时刻明智地选择合适的方法，从而有效地利用时间。

浮潜是一项很好的技术，项目团队可以用它来弄清楚需要研究多少领域，以理解问题和达成目标。往往在项目或者子项目的起始阶段，团队通过浮潜识别出研究范围、目标、利害相干人、研究边界、已知事实以及需要进一步做水肺潜水的地方。

当潜水员感觉到一些有趣的东西、陌生的事物或者更深的细节信息需要考察

① 多亏了 Dines Hansen，他让我们想到了这种表述。

时，他会进行较深的水肺潜水。深度潜水的发现常常会改变浮潜阶段所做出的假设。假如我们发现了某种海产生物，而此前我们并未预料到能在这片水域找到这种生物，那么我们就需要调查更宽广的范围，找出这种生物的繁衍区。

本模式的迹象之一是团队在做广度考察（浮潜）的同时，也会——而不是忽略——针对特定问题进行具体的工作（水肺潜水）。关键是团队在项目的整个过程中应用广度与深度研究技巧的能力。研究的广度范围识别出可能会对项目产生影响的人、组织、硬件和软件系统。在广度上知晓得越多，就越能识别出高风险与高收益的领域，以及如果辅以进一步的深度研究可能大有裨益的领域。

掌握浮潜与水肺潜水技能的项目团队不会因问题域之宽而气馁。团队成员知道自己不需要对整个问题域都进行同样深度的研究。例如，如果他们决定针对问题的某一部分购买已有的解决方案，那么他们的研究只需要深入到验证解决方案是否适用于工作情形即可。如果他们决定开发自己的解决方案，那么他们需要判断这项变动要求进行多深的研究。他们同样知道某些更深入的研究可以推迟到以后某个更为合适的时间。面对较宽问题域的项目团队在响应变化上面更胜一筹，因为他们可以预见改动所引发的影响。他们对于哪些是自己所知道的、哪些是自己所不知道的、哪些是需要加以探索的以及哪些是可以放在一边的，都心中有数。他们能够计划如何使用资源以获得最佳效果。

引入浮潜与水肺潜水技术的项目很可能把软件原型、模拟物与上下文建模联合在一起使用。他们也可能使用增量式的交付方法，在项目早期交付价值最高的功能。同样，他们也能够只用一页的篇幅就清晰地解释项目的范围和目标。

本模式的反例是团队要么沉迷于细节（“我们只做水肺潜水——懦弱无用的浮潜免谈”），要么惧怕细节（“我们是浮潜潜水员——换句话说，大海深处有怪物”）。并且，人们谈论起“更高层次”和“细节”就好像它们是独立的东西，与正在从事的项目无任何关联，这也属于本模式的反例。

优秀的开发人员不会画地为牢：他们既可以浮潜，也能水肺潜水。他们依据自己需要考察的对象来选择技术。侦察的时候，浅层潜水就已足够；但如果审察，就需要更深的潜水了。

有时，仅仅用脚趾探探水就足以知道能不能跳下去了。

模式 43　一切都是该死的接口



项目团队成员毫不妥协地强调接口——既在产品里面，也在人与人之间。

为了设计一个系统，我们必须了解系统与运行环境的接口。我们需要了解系统的原始输入和最终输出。在得到针对输入与输出的调查报告之前，我们只是做初步的分析：我们并不对问题划定范围。一旦拿到了报告，我们就能开始定义系统的功能。

在就功能达成一致的意见之后，又会如何设计呢？我们将大型的、复杂的系统切分成子系统，子系统又切分成组件。是的，一个有效划定这些子系统和组件范围的方法是依次枚举每一项的输入和输出。

我们如何分解实现的工作呢？通过子项目与/或通过组件。一个小组也许负责开发整个子项目，而个人则构建、测试组件。子系统和组件的范围定义了工作的边界，即每位开发人员的具体职责。这些接口是组件之间的协议——相当于某个组件要求另一个组件："你要分毫不差地给我这个数据——只有在这种情况下——我才会准确地创建这个产品，保存到那个特定的位置。"

在项目的早期阶段，在钻研得出所有详尽的细节之前，定义具备可实现精度的接口可能非常困难，而要意识到不被注意的细微之处就更难了。很显然，留下未定义的接口于事无补，因此我们必须根据当时最好的理解定义每一个接口。

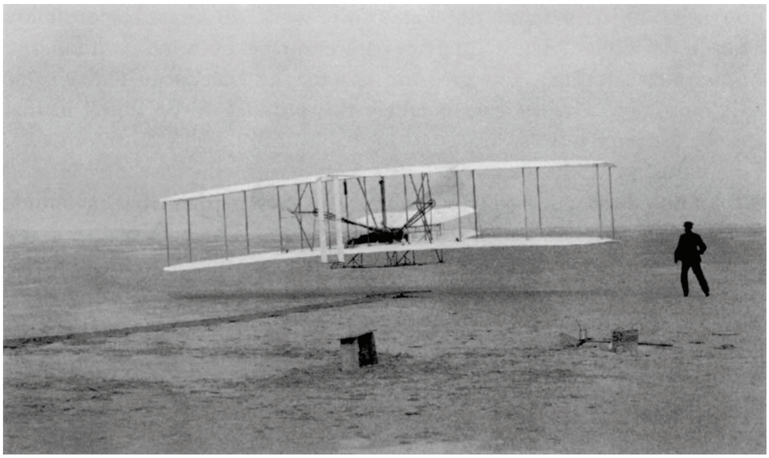
基于上述种种情况，接口极可能会出现问题，从而影响到至少两个——或许更多的——组件，而且这些问题通常也最难处理。

洞悉该模式的团队会早早地对付接口。在提交所有的组件代码之前，他们构建了一系列的程序来检验接口。他们早早地集成个人代码，频繁地测试。

> 我们曾遇到过一个项目包括了三个工作组——一个在加拿大，一个在美国，还有一个在以色列。经理在项目内部网上放了一份被他称为“接口圣经”的文档。那是唯一一份记录了所有系统接口的文档，其他有关接口的东西都不在其列。他极其信赖他的“圣经”，因此他不用再破口大骂那些该死的接口了。
>
> ——TRL

洞悉该模式的经理会警惕项目团队中的接口，防止出现任何一个工作组在任何一个接口上做出不恰当假设的可能性。请记住康威定律（Conway's Law）：产品反映了制造该产品的组织结构。对于接口，这一点尤为正确：项目中复杂的人类接口容易导致复杂的产品接口。

模式 44　蓝色区域



“奥威尔·莱特可没有飞行员执照。”

——Richard Tait，Grand Poo Bah，Cranium

团队至少有一位成员经常性越职。

让我们来见见温斯顿。温斯顿具备某种典型性格，而你不时会在开发项目中遇见这种性格的人。他并不是纯粹的无政府主义者，但看上去他只顾自己的想法。他似乎从来不去想自己这样做会不会有被解雇的危险，就做了很多自认为最有利于项目的事情。但是，他从来不会做得太过分。他只是越过了自己的权限——有时也超出了上级经理的耐心——到快接近于破裂的程度。温斯顿游走于蓝色区域。

经理在分派任务的时候会设定任务的范围：一方面要考虑到团队成员的能力，另一方面也保证了接受者拥有足够的自由度来完成任务。同时，经理也避免分派的任务产生重叠或者冲突。

考虑周详的任务界定给团队成员筑造了自由发挥的空间，接受任务的团队成员可以天马行空自由挥洒。但是，要精确地限定任务的每一部分工作几乎是不可能的。我们可以把项目任务的分派想象成创建了以下三个权限区域。

- 绿色区域由任务安排之中明确定义的所有事情组成：这是待完成事项的核

心部分。

- 红色区域包括了被明确排除在任务范畴之外的所有事情。
- 蓝色区域则包括了其他的所有事情：任务安排之中既没有明确要求也没有明确禁止的活动。换言之，它是处于绿色区域与红色区域之间的所有事情。

我们的同事温斯顿认为他可以做一切没有被明确禁止的事情。他不仅会完成被安排的任务（他的绿色区域），而且也会觉得自己有必要处理蓝色区域的事情——在他看来，这对获得最佳结果是非常必要的。他唯一的行动准则是自己所做的一切必须有益于项目。他不会等待许可，也不会去向其他人请求，他只是完成自认为需要完成的一切事情。

关于温斯顿的故事还有很多。我们有时甚至能看到他试图说服团队领头人同意他处理红色区域的事情。他只有去做那些被明确禁止的事情时才会请求许可。

团队里面出现了温斯顿简直让团队如获至宝。虽然有他的日子会惊险不断，但他能把事情完成得分毫不差。他的冒险天性意味着较之原来安排任务之时所预想的结果，他总能得到更好又更富有创造性的解决方案。

当你拿温斯顿与他的对立形象本森进行对比，温斯顿就显得尤为可贵了。本森是那种中规中矩的团队成员，他认为只要没有被明确要求去做某件事情，他就必须得等到许可之后才会动手。对于本森而言，蓝色区域就是彻底的“禁区”。无论进入“禁区”的价值有多高，他都不会去冒一点儿风险，除非取得了明确的许可。

因为曾经被告知不要冒险进入红色区域，本森把这种禁令视为永久性的、不容质疑的。他甚至可能站在一边观看项目失败，也不会建议采用某个处于红色区域的解决方案——那远远超出了正式安排给他的任务范畴。

本森与温斯顿的身上体现了下面这段似非而是的哲理：绝对的服从可能是有害的，某些善意的无序反而是有益的。

> “项目之中合适的无序度绝对不是零。”
>
> ——迈克·马舍特（Mike Mushet）

模式 45　消息美化



坏消息在组织里面没有准确地向上传达。

在有些组织里面，坏消息从来不上报。更为普遍的是，坏消息在一级一级往上传达的过程中被美化了。想想上面展示的例子。

消息美化是一种破坏型模式，因为它使决策者得不到所需要的信息，这又可能导致错误的决策（或者贻误了决策），结果反而更糟。如果信息流动更为有效，那么许多众所周知的错误决策原本可以避免。“挑战号”航天飞机于 1986 年 1 月 28 日起飞的决定也许是本模式最惨痛的例子。

Diane Vaughan 在她的著作 *The Challenger Launch Decision*[①] 中透露，因为担心固体火箭发动机各段之间的 O 形密封环在寒冷天气下的工作性能，来自 Morton-Thiokol 的工程师建议取消起飞。在遭到马歇尔航天飞行中心的官员批评之后，Thiokol 的高级经理撤销了这些工程师的建议，同意了这次起飞计划。事实上，Thiokol 最初的鉴于天气寒冷而取消起飞的建议并未由马歇尔的经理传达给高级 NASA 计划经理。在罕见的寒冷天气中继续起飞的决定导致了宇航员的死亡与航天飞机的毁灭。

谢天谢地，并非所有的消息美化都会导致如此悲剧的结果。当你在项目中观察到这个模式，最典型的症状是出现意外，典型的后果就是延误了一个又一个截止日期，从而达不到期望的结果。

---

① Diane Vaughan，*The Challenger Launch Decision: Risky Technology, Culture, and Deviance at NASA*（Chicago：The University of Chicago Press，1996）。

项目中的意外通常伴随着与下面类似的故事。在几个月的开发之后，伴随着中途顺利的交付，新系统进入最后的测试环节，并将在一个月之内交付。在开完针对最后阶段开发活动的计划会议之后，项目经理报告需要额外一个月的工作时间，系统才能做好交付的准备。勿需多言，高级经理听到这些都惊呆了。团队怎么能在快到计划截止日期时才发现自己无法交付？

司空见惯的答案是团队中的很多成员其实很早就已经知道了截止日期并不现实。他们可能向自己的上级反映了这个问题，或者甚至在他们的进度汇报中表达了自己的担忧。但是在项目一线经理到高级经理之间的某个环节，他们对于项目计划安排的怀疑并未反映在项目的态势报告之中。

刻意隐瞒坏消息可能使得可解决的问题变成无法解决的问题。那些原本能对意外的延误有所作为的少数人——控制着资源、给项目设定外部期望的高级经理——得不到采取纠正举措的机会。等到整个过程的后期，所有纠正举措都已不再有效。如果他们可以在足够早的时候就知道待完成的工作与可用的资源和时间之间的关系不对等，他们就能在足够早的时候提供更多的资源，或者重新调整范围，或者延长计划时间，以避免在最后关头出现延误。或许即便他们提前知道了坏消息，也不一定能*真正地*解决问题，但如果他们从未听说过这个问题，他们肯定*无法*解决它。及早的警报非常关键。

那么，为什么会出现这个模式呢？最普遍的原因是恐惧。没有人喜欢听到来自于自己所关心的人或事物的坏消息。非常普遍的是，经理让这种人之常情的个人厌恶影响到他们对坏消息作出的反应，甚至更严重地，影响到他们对那些报告坏消息的人的态度。针对这种行为，我们会念咒语“不要射杀报信人”，但那往往不管用。如果组织的管理文化——更多是通过它的行动，而不是言语——很清楚地表明：送来坏消息的人会吃到苦头，消息美化就变得不可避免。

关于管理层引起的消息美化，至少还存在另一种类型。团队成员早在他们能够*证明*之前，就*知道*项目陷于困境。在有些文化之中，声称达到预定日期有问题的团队成员很有可能会遇上那个臭名昭著的质疑：“你怎么这么肯定你们*无法*做到？”由于不想被视为悲泣者或者懦夫，团队成员就此缄口不语，直到灾难已经无可置疑（而且通常也是无可避免）[①]。

① 可参阅第 46 项模式。

怎么才能改进组织的能力，使之能快速、准确向上传达坏消息呢？大部分的解决方法在于你——经理。你不能仅仅在口头说希望即时得到坏消息，你还得那样去做。最起码地，这意味着你得把对坏消息的反应分成两部分：(1)决定如何处理；(2)弄清楚它是怎么发生的。首先关注第一部分。不要马上清查为什么坏事情会发生。相反，把精力放在督促你的团队上面（包括坏消息的报告者），提出“改进”计划，然后落实到行动。你非常注重富有成效的纠正措施，这样你的所做所为就不大可能被团队成员视为对他们的指责或者惩罚，因而也不大可能导致人们将来对坏消息进行隐瞒或者歪曲。最后，你一定要分析问题的根源。这样，类似的不幸才能被避免，但这可以等到事态已经发展良好了再说。到那个时候，人们的防戒心理往往更少。坏消息可以被解决，但不能被美化。

模式 46　慢慢地道出事实



公司文化迫使人们把令人不安的消息埋在心底。

1994 年，比尔·克林顿总统任命迈克·麦克寇里（Mike McCurry）做他的新闻发言人。当时，麦克寇里早已在记者团中众所周知。在他的第一次记者招待会上，一位记者问他："作为新闻发言人，你不会向我们说谎了，对吧？"麦克寇里回答道："是的。但是，我将慢慢地道出事实。"

项目经理与团队成员有时也会出于下面的原因，结果慢慢地把事实道出来。

- 他们不希望对自己公布的问题负责。许多文化都会传达出这种信号，那就是——如果你发现了杂乱不堪的现象，你就得负责清理。

> "哎呀，老板，在把原有主干系统的并发用户增加到三倍之后，我们现在可能有大的性能问题。"
>
> "你说得对，Smithers。你来处理那个问题吧，不要让它再次出现了。"

- 他们对自己将会被问到的后续问题没有答案。指出问题，却又不立刻提出改进措施——这会被认为是在发牢骚。而在很多组织里面，发牢骚者的职业生涯是有限的。

  “哎呀，老板，这个项目可能会延迟。”
  “多晚，Smithers？”
  “哎呀，老板，我不知道。”
  “又是发牢骚，”老板压着嗓子嘀咕道。

- 他们在等待其他的人揭露更大的问题，这样他们就能隐瞒自己的问题。这个教导来自于“总有人会先退缩”职业发展中心。

  “我这次把项目所有的小组组长召集在一起开会，是因为 Smithers 告诉我他的小组比进度安排表至少晚了两个月。但他并不是非常肯定，”老板叹息着，“于是，为了安全起见，我把这次交付正式往后推迟了 4 个月。”

  小组组长们大声喊道：“那太糟了，Smithers。不过，我们可以利用这段额外的时间来进行更多的测试了。Smithers，你需要 Ralph 小组的新人帮助你么？”

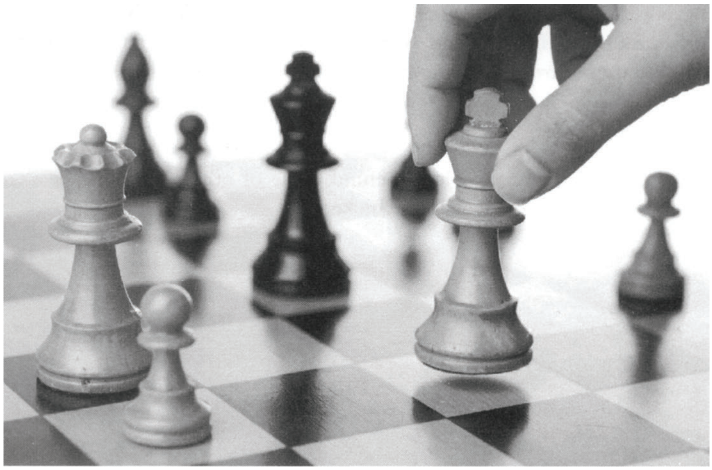
  这是那些小组组长们表面上说的话。他们的真正意思是：

  “Smithers 站出来了！我们就知道他是一个胆小鬼，爱发牢骚。希望这延长的四个月能掩盖我们自己所不为人知的延误。”

如果 Smithers 能在位子上待上足够长的时间，他将明白，迅速地揭晓真相是一条殉难之路。他也将会明白，组织不希望听到即时的事实，而希望尽可能长时间地保持其乐融融的和谐局面。他将发现即使事情真相最终会被披露，但组织倾向于等到那一天来临再“着手处理”。而在那一天之前，Smithers，作为团队一员，需要慢慢慢慢地把事实讲出来。

把自己的问题隐藏在其他人的问题之后，这种人有时被称为“进度胆小鬼”（Schedule Chicken）。

模式 47　残局游戏



团队在整个开发过程中定期地使用交付标准检验构建中的产品。

这是你新学期的第一天。你的老师在欢迎了你和其他的同学之后，就开始讲述他计划如何指导这门课程。最后他这样说："我并不认同等到学期末再最后进行测验的做法。我更愿意在整个学期里面，每两个礼拜就做一次测验。当然，每次的测验肯定不会一样，但是它们都会以大同小异的方式涵盖相同的内容。"

这听上去也许有些古怪，但它却与我们所讲的"残局游戏"非常相像。

为了理解这种做法的合理性，让我们先将其与绝大多数软件项目的做法进行比较——就绪度审查者的舞会[①]。这种做法按时间顺序常常包括了下面这些活动。

- 定义交付标准过程中的拖延："呃，离交付日期还有几个月之遥，就绪度审查就定在计划交付日期之前一个礼拜左右吧——现在还有真正要紧的事情等着做呢！"
- 预审查演习："噢，我的天，距离就绪度审查只有不到两个礼拜的时间了！我们把那些交付标准扔哪里去了？我们该怎么测试这些标准？天啊，我们

① 此处原文为 Readiness Review Folk Dance，作者把某些软件项目的就绪度审查比喻成审查者的"舞会"。——译者注

怎么才能检验这个？”

- 严肃的就绪度审查仪式：每个人都知道就绪度审查离交付日期是如此之近，任何重大的问题都会导致团队错过交付日期——“已经没有足够的时间来认真修复缺陷了”。因此，每个人都意识到唯一可被接受的声明就是“已准备好交付”。太多的缺陷？有升级包！
- 审查之后的灾难演习：在审查仪式声称顺利完成之后，少数人忧心忡忡地聚集起来，试图找出哪些与交付标准差得最远的部分可以在交付日期之前完成，而不会带来更糟糕的影响。
- 无人汲取教训之叹：没有人喜欢汲取教训。每个人都信誓旦旦下次会做得更好。不幸的是，下次又是在几个月以后。而我们又必须开始下一个版本了，因为现在还有真正要紧的事情等着做呢！

就像大多数民间舞蹈一样，这些情况随着团队文化的不同而存在差异，但相信你已经根据这些描述识别出了自己团队的一个或两个方面。

采用“残局游戏”方法的团队在项目早期就制订了交付标准，然后开发出验证产品是否满足交付标准的测试。这些测试在每次开发迭代结束之后都会执行，然后是一个小型的就绪度审查会议。

考虑一下这种做法带来的如下好处。

- 在开发的每个阶段，你的团队都会再一次关注到产品交付的剩余工作。
- 一旦此前已经通过的回归测试失败，你就可以及早得到警报。
- 随着项目进行，你有大量的机会来改进交付标准。

这种流程也许令人觉得别扭。一些交付标准在开发的早期也很难被检验。但无论如何，即使只是那些大写的黑体“TBD”（待完成事项），也能引发出有意义的问题，比如“嗨，我们什么时候会第一次运行性能测试套件呢”。

模式 48　音乐制作人



在 IT 组织里面，拥有音乐才华的人所占的比例超出了在平常群体中的比例，有时甚至还会大很多。

我们承认自己不知道现代社会有多少人能够演奏乐器，所以我们就在黑暗中歌唱[①]（故意的双关）。但是，我们坚信能够发现一种模式：在信息技术行业，音乐家的数量多得出乎寻常，在有些组织里面我们可以很明显地看到他们聚集在一起。

在每一年的课程结束之后，我们每一位 Atlantic Systems Guild 的员工都会联系大量的 IT 组织。不经意的提问总能发现音乐人士在被调查人群之中的数量多得出乎意料，其数量之多远远超过日常生活中能遇上的音乐人士。这或许缘于音乐的数学与逻辑基础性，以及音乐对于技术型思维的人士非常具有吸引力。技术的数字本质与音乐的模拟本质之间的对比可能是非常地奇妙，又抑或只是个巧合。

我们也发现一些组织充满了大量的音乐天才。下面的故事具有相当的代表性。

> 我最喜欢的例子是 Landmark Graphics 的年度软件大会，在大会上所有的项目人员聚在一起谈论他们在过去一年中所做的事情。会议时间

① “在黑暗中歌唱”（*Singing in the Dark*）同时也是一部电影的名字，有兴趣者不妨上 imdb 了解更多。——译者注

并不是全由大会委员会主导，有些是属于音乐、歌唱以及舞蹈的，而且都是由雇员主导的。宾馆中处处都是乐队，如果你没有参与其中，你可以漫步在走廊上寻找自己喜欢的音乐。乐队都棒极了。一些 Landmark 的音乐师告诉我们排演起来有时会很困难，因为鼓手在卡尔加里，而其他的乐队成员在休斯顿。尽管困难重重，他们还是想方设法聚在一起，在年会的盛大之夜演奏出美妙的音乐。

——TRL

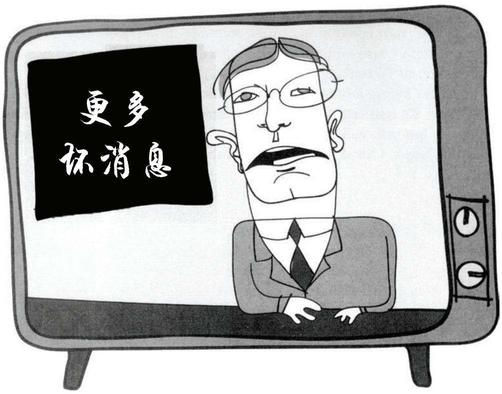
来自 Infovide-Matrix 的 Borys Stokalski（本模式插图中用麦克风演唱的人正是他）告诉了我们下面这个故事。

“夜已经很深了，可 Infovide-Matrix 公司里面依然灯火通明。有人也许以为又是一个悲惨的故事：追赶着不可能的进度，或者制作第二天需要向重要客户演示的幻灯片。但这个故事却充满了快乐、友谊，以及……一些嘈杂。Wojtek，某银行的 IT 主管，在捣腾着他的鼓。Lukasz，活泼的软件质量顾问，在他的贝司上面试弹着一些乐段。而 Pawel 这位优秀的项目经理，在给他喜爱的 Gibson Les Paul Supreme 牌电吉他调音。Aidan，一位大软件厂商的联盟经理，尚未抵达。一旦他到了，他的中音萨克斯管就会给他们今晚将要尝试的乐曲加上一些美妙的旋律。Grego，IT 治理能力中心经理，跟我讨论着吉他改编——舞台上出现三个吉他手的话，就需要一些筹划以避免声音混杂。只要有小型爵士乐演奏会，我们就会凑在一起练习、表演——为了纯粹的音乐与舞台表演的乐趣。这就是下班后的摇滚乐。”

在硅谷里面，身为 Oracle 开发部门副总裁的 Vittorio Viarengo 同样领导着一个小型的爵士乐团 Jam4Dinner。该乐团全部由软件从业人员组成，他们既表演三重奏也表演五重奏。这个乐队的一些唱片还在自己的网站上有售。

试着在你的组织中四处打听一下，看看多少同事也是音乐人士。我们无法保证你能找齐一支完整的管弦乐队，但你很可能可以组建一只弦乐四重奏乐团或者摇滚乐乐团。我们的 iPod 等着你们的音乐。

模式 49　记者



记者是指那些把准确报告这个目标与让项目成功这个目标完全分开的项目经理。

项目经理知道他们必须理解项目的真实状态，然后基于那些情况准确地做出报告。然而，有时他们不明白为什么要如此细致入微：保证项目达成目标。他们把始终准确地报告项目状态作为自己的目标。实际上，他们变成了——记者。正如影评人①那样，项目记者坚信——但愿仅仅是潜意识中里——即使项目失败，他们个人也能成功。

想一想记者在报道飞机失事时的情景。记者觉得自己有责任准确地报道哪架飞机失事了、何时何地、飞机上有多少人，以及是否有人生还，但他不会因为没有阻止飞机的失事而觉得内疚。那是别人的工作。

记者型项目经理也同样如此。他们的报告条理清晰、准确到位，而且细致入微，可以列为范本。他们清楚地知道“订单条目”子系统延迟多长时间、偏离关键路径多少天、这项延迟将会如何影响依赖于斯的下游任务。但是他们却忽略了一件非常重要的事情：他们的角色之所以*存在*，是要保证他们的项目有个圆满结局。正如飞行员的首要目标是避免任何一位乘客的伤亡一样，项目经理首先要保障项目安全、准时“着陆”正确目的地。随之而来的准确报告只是达成这些目标的一种手段，但却无法代替这些目标。

① 请参阅第 19 项模式。

模式 50 空椅子

没有人为整体用户体验的概念一致性负责。

> 从前，我与一家开发安全系统的公司有过合作。他们的新一代产品计划支持语音输入/输出，并在其小型设备系列上支持触屏界面。因此，管理层创建了两支用户界面团队，一支团队负责触屏，另外一支则负责音频接口。这两支团队位于不同的城市，埋头完成各自的特性列表，期间甚至都没有询问过设备将支持怎样一个整体业务流程。从外部来看这个项目，谁都能马上看出如果两个团队之间进行更多的讨论，他们会更好地使用这两项技术。
>
> ——JSR

假如公司签获了一个新的开发合同，你被任命为项目经理。你自信可以胜任这项工作，因为你拥有所需的全部技能和经验。你按照工作所需技能的不同，在公司的不同部门之间，或者在你所管辖的部门与合作公司之间分派工作。你下面的子项目经理都非常称职，他们以饱满的热情接受了任命。



子项目团队对分派的任务非常满意，他们明白项目的整体目标，而且在各自的领域上都做得非常出色。倘若不是为了争取更多的时间、金钱或其他资源，他们从不来打扰你。你的子团队工作于不同的地点，但是你一点也不担心，因为你已经很清楚地记录了各个团队之间分派的任务。他们按照程序要求相互合作、沟通接口，并分享各自的中期成果。

客户在项目上投入了一流的领域专家，但在不同的子项目上都安排了不同的人。有时候，这些领域专家甚至都不知道

他们的同事中有谁也在参与同一个项目的其他部分。

你领导着整个项目，与来自客户组织的高层经理一起紧密合作，调整预期目标，跟进众多子团队的进度。

但是，你的项目所生成的产品，几乎肯定不会得到最终用户的好评。哪里出问题了？

你的项目中空了一把椅子。很多项目没有真正成功，只因为缺少一个人专门负责确保最终的业务流程——从用户的角度看来——尽量顺利地开展。这个人关注的是整个项目对于客户而言的最佳的结果，一直到最细微的细节。

我们说的这个人不是项目经理，也不是项目团队的总领导。这个人可能没有任何人直接向他汇报，他也几乎肯定不需要为预算或者计划负责。他全部的关注点都只在于产品如何与目标环境交互，特别是与产品的最终用户交互。

这样的人员被赋予了各种各样的头衔：产品经理、系统架构师、业务分析师等。有时他们自称为技术项目经理，因为他们的工作在于关注解决方案的细节（相较于那些处理预算、人员安排和计划的总体项目经理）。无论头衔如何，这些人不属于任何子团队，他们的工作贯穿在所有的子团队之中。

就算只是在客户那边存在着这样一个人，那可能也已经足够了。如果来自于客户组织的某个人不断地询问子项目的协同状况，即便承包商这边缺乏类似的统筹角色，你们或许也能成功。

空椅子在那些需要集成原有系统的项目中甚至更为常见。往往只是集成工作的技术因素在驱动着项目，而业务集成的细节、用户交互的工效因素，以及可能产生突破性创新的想法都被忽略了。

环顾一下你的项目团队桌子，是不是空着一把椅子？

模式 51 我的堂兄文尼[1]



团队成员争论不休——群情激昂却了无敌意——去评价和改良他们的主张。

人类自从有了语言之后，就开始了争论。在那之后，并非每一次的争论都有好处，在很多情形中，争论者实际上并没得到多少好处。尽管如此，从整个历史长河来看，怀着良好意愿的人们通过争论和辩论来验证主张——同时也在不断地改进这些主张。

无论何时，团队成员都会因自己的主张和建议而争论。他们把争论当成一种探讨各自意见、达成一致看法的手段。如果对于某项主张的论辩无法说服其他人，那么其他人很难会接受这项主张。但是，一旦持怀疑态度的人被论辩所说服，他们基本上就会不遗余力地拥护该项主张。如果在论辩的过程中发现了某项主张里面的小瑕疵，请耐心一点，队友通常会争取修正它们。在争辩某项主张的过程中，针锋相对的争论几乎总能导致新想法的产生。

争论的关键在于说服别人，而且在这样做的同时，我们也在说服自己。如果

① 美国 1992 年的一部喜剧片。讲的是纽约大学学生 Bill 和 Stan 在度假时被误认作是谋杀犯抓了起来，并要接受审判。而他们没有足够的钱请律师，Bill 就只好找来他的堂兄 Vinny，一个毫无经验的新手作他们的律师。——编者注

你想要说服别人，那么理所当然地，你的主张必须组织严密、表达清楚。换句话说，你不得不多加考虑，想想自己的主张能否经受住随之而来的疾风暴雨般的——而且无所遁形的——审查。我们都希望在争论的时候表现得知识渊博，因此，就得更加谨慎以确保我们提出的主张合情合理、论据充分。

本模式因之命名的男主角通过机智的辩论赢得了他的诉讼案件。他对案件的阐述以及他对控方的质询都足以打动陪审团。类似地，对项目有益的争论并不是一般意义上的口舌之争，也不是在大部分办公室里上演的口水战——诸如谁支持的足球队最厉害、Mac 和 Windows 哪个更好等。有意义的争论能够改善构建中的系统。什么样的设计最契合需求？何种程度的安全级别既为保存的信息提供了最佳安全性能，又满足了访问所需的级别？再有，鉴于已授权用户的意外错误操作比外界的代理入侵更为普遍，防止前者的发生是否比防止后者的发生拥有更高的优先级？正如项目团队遇到的很多问题，这些争论是多方面的，需要公开陈述、辩论，直到得出最佳结论。

有些争论牵涉了很多方面。假定争辩的人们正在决定产品的整体观感，则市场部的人们会为炫酷的、整齐如一的界面而争辩；可用性专家则会为足够的可视化控件（使得一般的工作非常容易完成）而争辩；开发人员则为自己钟爱的特性而争辩，反对将会导致不优美实现的一切事情。

> 有些争论只是小规模的，但是依然十分重要。在争论如何减少磁盘访问指令数量的时候，我坐在那里茫然不知所措。你绝对未曾见过，争论者们都坐在自己的办公位置上，隔着工位挡板发出争辩声。
>
> ——JSR

为得到最佳解决方案而争论的团队成员会互相尊重，甚至说他们惺惺相惜也不为过。否则，他们不可能富有成效地进行争论。当争论的时候，团队成员知道对自己主张的讨论和剖析并不是针对个人的攻击，而只不过是想通过有效的方式交付最优秀的产品。但是这种安全感并不是来自于仁慈的管理层，抑或好心的团队领导，而是因为团队成员知道争论不是针对个人，也不是建立一种权势等级，更不是展示令人讨厌的个人知识优越感。它来自于“其他人是自己的堂兄文尼”的认识。他只是在检验你的主张，并通过与你争论的方式试图改善你的主张。

模式 52　特性汤

“体面汤、浓又黄，

盛在锅里不会凉！

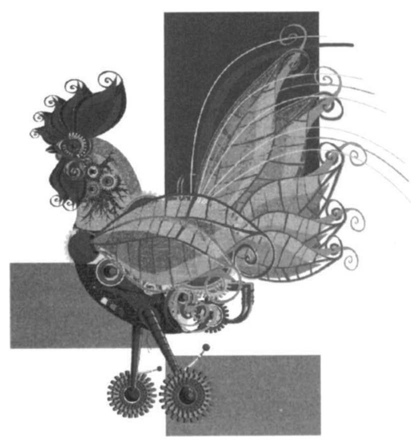
说什么山珍海味，哪儿有这样儿香？

半夜起来喝面汤，体面汤！”

——刘易斯·卡罗尔（Lewis Carroll），《爱丽丝漫游仙境》①

产品夸耀自己繁多的零碎特性，其中很多对于解决客户真正的业务需求几乎毫无帮助。

在刚开始的时候，一切都显得那么美好。市场部接到一个客户的请求，要添加额外的下拉菜单。然后，又有人要求在产品中添加一个输出接口，产品经理想要加上一份新的分析报表，DBA 要求在数据库里增加一个新字段并改变背景的颜色。所有这样那样的需求，都交由开发人员负责加进到产品里面。随着需求的不断添加，产品的特性集不断增多，但过了一段时间之后，每个人——市场部、客户和开发团队——在如何将所有这些碎片整合在一起以及如何利用它们实现业务目标这些问题上，开始变得盲目起来。曾经带着明确目标出发的项目变成了难以下咽的、由各种无关特性炖成的一锅汤。



情况变得更加汤汁淋漓，因为各个利益相关方都从不同的角度来看待产品的需求，根本不存在共同的、连贯的思路。市场部从营销的角度把需求打包成一组一组的特性集合，也不管它们在功能上是否内聚；开发人员则按照自己所使用的实现技术对需求进行归类；各个客户也只是从他个人工作的角度出发单独地对需求进行考虑。这些离散的需求所带来的后果就是每个人谈论进度或者对变更做出

① 译诗引用的是赵元任先生《阿丽思漫游奇境记》（1992）的译文。——编者注

决定的方式都不一致。按照发布产品版本的主题再取折中已不可能，因为根本就不存在一致的主题。相反，产品变成了混杂着各种玄机的大杂烩。

为什么如此多的产品最后都沦为特性汤了呢？始作俑者还是需求的源头：人。

人们会自然而然地认为自己的需求才是最重要的。同一个组织中的不同部门或者不同的客户，都想获得属于自己的、与众不同的特性，于是提出的需求根本不顾及产品在整体业务上的一致性也就不足为奇。而这正是分析师的工作。

当零散的需求来了之后，分析师需要将它们与其影响的业务流程映射起来。这种映射提供了一种方法——向不同的人们展示变更需求会对他们的工作产生哪些影响（这些影响有时非常令人惊讶）。这种分析让分析师获得了基本的理解，从而进一步发现人们真正需要的是什么，以及变更是否提供了真正的好处，抑或仅仅是另一个滴入汤中的特性。

特性汤的另一个来源是设计人员在面对一项新需求的时候，不去追究其与既有产品在整体上有何关联，就将它加入进来。设计人员应该先问一问："它是否属于已声明的范围？它与既有产品的接口是什么？它是否重复或者搞乱了已经存在的东西？"

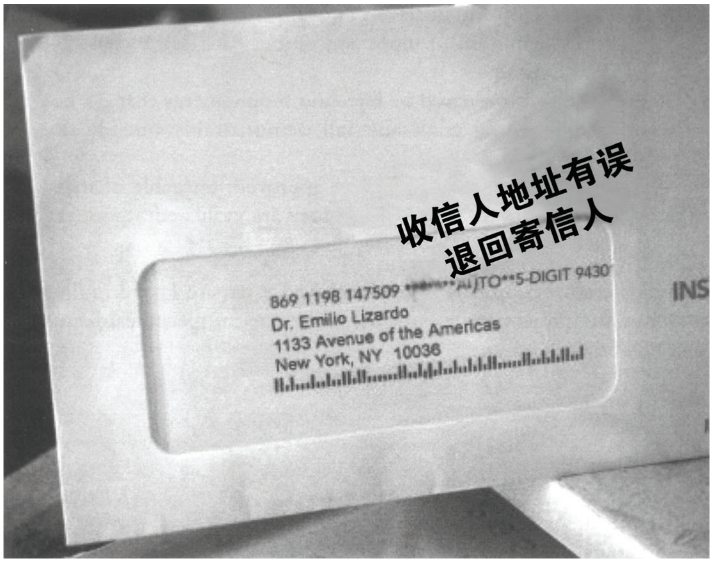
在解决这些问题上的不断失败导致产品变成了一堆离散碎片的组合。基于这种离散的特性的需求从本质上来说，意味着项目对于"什么是属于范围内的"以及"什么是超出范围的"没有客观的定义。因此，额外的需求就很容易从不同的来源渗透进产品里面——事实也确实如此。产品变得越发分崩离析，它也就越发难以评估，做出的变更就越发难以前后一致，一路螺旋直下，回天无术。

避开特性汤的组织有着很多的共同点，如下所示。

- 尽可能干脆、尽可能早地定义项目目标和非项目目标。
- 声明项目范围，并以精确定义输入/输出数据的形式时刻保持更新（参阅第24项模式）。
- 坚决地拒绝那些对声明的目标没有积极效应而又明显超出项目范围的需求。
- 新需求的添加遵照被核准的、可追溯的变更管理流程进行，同时使用项目声明的目标对它们进行评估。

避免特性汤得靠训练。时刻牢记着是你们整个项目团队，而不是零散特性的请求者将会身陷浓汤，培养这种意识绝对是值得的。

模式 53 数据质量



数据质量经常会糟糕透顶。遗憾的是，解决这个问题的常见做法是寻求更好的软件来处理数据。

数据库软件的质量超出它所处理的数据的质量，这并不罕见，然而在最终用户看来，系统的质量受制于上述两者之中更差的那个。每个公司的数据库里面都充斥着不准确以及过期或缺失的信息。问题就像鼻子长在脸上一样明显，但每个人要看到自己的鼻子却非常困难。即便每个人都能看出其他人的数据问题，公司也很难去直面、应对他们自己的数据质量问题。相反，公司看到的总是软件与数据合在一起的问题。因为软件总是比数据（数据量多得可怕）更易于修复，所以公司选择去修复软件，或者对之更新换代。

这些都意义不大，因为我们要讨论的关键问题不是“为什么我们不应该那样做”，而是“为什么尽管不应该那样做，我们却仍然那样做了”。部分原因是出于“消息美化”（请参阅第 45 项模式）的特殊情形——坏消息（比如这个月的发票有 2.4%都被因为无法送达而退回）在向上传达的过程中，在每一层中都遭到了愤怒的质疑：“你究竟要对这个搞什么？手脚麻利点！”

这个“手脚麻利”立刻排除了大范围手工修复的可能性。含糊的回答是，将会立即开展重大的“数据清洗”工作。“数据清洗”这个有吸引力的短语在送到 CEO 级别的人物面前时已经意义全非。在组织层级的最底层，数据清洗意味着拨打电话或者连上因特网，仔细查阅相关的文件，研究和纠正每一份单独的错误数据。在上层，这意味着更为巧妙的方式——用某种方法很聪明地处理错误的数据，理清并得到正确的数据。由于资金来自上层，分拨的资金往往放在了更为巧妙的方式上面，而不是由大量的书记员完成真正的工作。

指出数据可能会被破坏（比如，因为错误的计算）是值得的，而且在这种情况下，至少有一些半自动化的方法能够通过恢复更早的备份版本来抵消造成的破坏。类似的是，当相同的数据各自记录在多个系统里面，一些自动化的数据清洗能够帮助把更优的数据分离出来。在上面两种情况下，自动化的数据清洗依赖于对冗余数据的利用能力。虽然很容易想到利用冗余数据达成数据清洗的例子（A 系统有一个老的地址值，而在 B 系统有一个新的地址值），但是可以自动化清洗质量低劣的数据的事例其实很少，简直是大海捞针。

数据质量随着时间推移而逐渐下降的主要原因是变更。对于我们称之为“公司数据”的资产变质问题，也许只能靠手工修复。空想着去采用别的方法只会拖延清算的日子。

模式 54　本

对于有些人而言，工作条件简直太好了，或者项目太有趣，又或者产品太酷，以至于他们对工作的热爱大于对薪水的热爱。

本——不是他的真名——在一家 CAD 软件公司工作。作为一名软件工程师，本对高等数学有着非常透彻的理解。除了自己日常的项目工作，本会对其他有问题却无法自行解决的人施以援手。（都是些非常困难的问题，毕竟，本的同事也都是非常聪明的人。）问题通常都不属于本的职责范围之内，有时甚至都不属于他所在部门的职责，但是他与这些同事一起并肩战斗，而且最后常常能找到满意的解决方案。



这则故事的重点是本从自己的本职工作之中获得了大量的乐趣。这个人从事了很多困难的任务，并且都成功了。他热爱自己的工作，接受工作带来的挑战，认为工作妙趣横生，并且绝对不关乎金钱。涨薪或者奖金当然欢迎，但它们根本无法激励本。而且，因为本对工作的兴趣比对组织的兴趣更大，因而他也不会仅仅为了涨薪而选择辞职去其他公司。

我们遇见了很多像本一样的人。在我们的咨询项目里面，我们会不时地与本一起工作。他们在组织中拥有不同职位，从事着非常困难的任务。他们并不总是团队之中技能最强的那个人，也不总是薪水最高的那个人。然而，通过他们脸上满足（但从来不会沾沾自喜）的神情和因为热爱当天工作所流露出的镇定态度，你能意识到他们的存在。

虽然本很容易管理（而且令人愉快），但更容易对他管理不当。一个惹人讨厌的经理在手下的员工离职之后不去招聘一个替代者，因为知道本是如此热爱本职工作，经理认为自己可以把更多的工作交给他。经理逐渐把工作的担子移到本的身上，但是一旦工作负载变得无法承受，本就再也无法感觉到工作的乐趣，进而

选择离开。而且，因为离开的是本，因此公司也失去了最好的员工。

经理失去的要多于本所失去的。本总能很快地找到工作，而经理要找到一位本就困难多了。

本不需要密切的督管。本的经理的角色就是引导本去从事他感兴趣的工作，从而保证他这个高度胜任并热爱这个工作的员工以饱满的热忱去完成它。

模式 55　礼数小姐



人们认为质疑同一个团队的成员的主张是不礼貌的。

在一些组织里面，任何批评都被认为是针对个人的，因而视作是禁忌。在某些情况下，工作产出与工作生产者被视为一体了。这种古怪的逻辑就像这样："批评梅格的模式就是批评梅格的能力，而这是对梅格个人的指责。我从来不会批评梅格，因为这有可能伤害到梅格的感情，同时其他人也会指责我批评了梅格。"

批评变得委婉，变成各种含蓄的说辞，比如述评或者评估。任何缺少"做得漂亮，哈尔"之类的述评会让房间里的每个人都觉得不快。

这种滥用礼貌的结果就是严重的平庸。工作无法得到真正的改善，无论以何种形式，完全重新开始工作或者推翻重写都是不可能的。没有人会说，"让我们丢弃这份代码，重新考虑一下整个前端"，即使那在当时的场景下是最恰当的做法。

错误的"良好"礼数来源于从组织某个高层传来的明确（但从未公开挑明的）信息。这是披着"礼貌"外衣的一种懦弱的表现。

"我们将一直尽最大的努力来礼貌对人"的规则在大多数健康的组织里面是有

意义的，也是受到认同的，但是在礼数小姐的组织里，这条规则下面还有一行小字：“批评永远是不允许的，因为一旦开始，它就会四下扩散，而我们的文化还没有强大到从这种反省行为中受益。由于无法证明决定到底是好还是坏，所以就应该接受所有的决定，不准有任何意见。”

礼数小姐的组织里面都是假面孔，没有真面目。身处其中的人们被迫整天带着面具。

模式 56　全神贯注



在单一的项目上投入全部的时间，可以改进个人的绩效。

德里克·雅各布（Derek Jacoby）在伦敦西区剧院上演的戏剧《破译密码》（*Breaking the Code*）中传神地扮演了阿兰·图灵这个人物，他全身心地投入到整部演出之中。他没有接受轮演剧目中的不同角色，而是把自己所有的注意力和才智都投入到这部戏里面。他学习图灵的处事方式，与其他角色一起排演，探究图灵的人生，决定如何才能最佳地刻画这位伟大的数学家、逻辑学家和密码破译家。之后，他在每个晚上都饰演这个角色。

戏剧界把成功授予德里克·雅各布这一类的演员（这些人每次都是全身心地投入到某一场戏剧之中）。我们发现一旦软件开发人员在单一的工作任务上面做出类似的投入，软件项目往往也会取得成功。

雇主聘用知识型工作者，是为了利用他们的能力和智力。假设你招聘了一位专家，他拥有最高每小时 100 单位的智力产出率。作为雇主，你千方百计地想让他以最高的产出率工作。你显然要照顾好他的物质享受，想办法把他的专注时间最大化，并创造条件为他提供各种帮助。然后，你把他分派在一个项目上面，使其全职工作，而他则集中精力、开足智力马达进行工作。假设他以最高产出率工作 40 小时——他就给你产出了 4000 单位的智力。你采取所有的手段从雇用的人才那里得到利润。

你从这项交易中获得的是全神贯注，这使得对单个项目工作的专注能带来最高的智力产出率。

被分派到多个并行开展项目上的员工不可能保持最高的智力产出率，因为多任务是要付出代价的，需要消耗一定单位的智力成本。从 A 项目的上下文切换到 B 项目，需要耗费一些脑细胞来了解 B 项目的状态。找齐所有正确的 B 项目文件，从大脑中消除 A 项目的思绪，重新与从事 B 项目的其他人员建立联系，把之前建立的思路重新建立起来——这些步骤对于大脑重新适应 B 项目的上下文都是必不可少的。

> “同时做两件事情，你就把你的 IQ 分成了两半，而且相信我，那些分掉的 40%真能产生差别。”
>
> ——Dale Dauten，《纽约时报》(2007 年 4 月 29 日)

在最乐观的情形下，产出的流失率被减到最小，因为你的员工们可以阅读条理清楚、前后一致的文档，来帮助加快重新适应的过程。但是，如果文档不齐全，而项目的工作人员却又很需要相关知识，那该怎么办？为了能够有效地工作，工程师需要清楚团队此前与客户、管理层之间的讨论、项目会议、未解决的问题以及众多其他方面的项目经历。除了重新拾起项目的产出物需要时间之外，重新与项目的其他成员建立人际关系也需要花费一定的时间。团队成员之间的日常联系产生错综复杂的共同经历，其效果就像凝聚剂。缺席或者无法接触到的团队成员稀释了这种凝聚剂的强度。

有些人试图量化研究背景切换的成本[①]，但因为其中涉及了大量的变量，估算起来非常困难。然而，基于在人类注意力与信息传输方面的观察，我们确信背景切换的确会导致生产率的巨大下降。意识到背景切换与生产率之间关联的组织会避免把一个人分派到多个并行的项目上面，从而提高了员工全神贯注的效率。

① 据杰拉德·温伯格（Gerald Weinberg）预计，当给一个人分派三个项目时，背景切换的成本是 40%。换句话说，他 40%的工作时间被无产出的任务消耗掉了，剩余的 60%被分割到三个项目上。以一周 40 个小时来计算，假设时间被平均分配到每个项目上，工人最后是在项目 A、B 和 C 上各花了 8 个小时的工作时间，还有可观的 16 个小时被花在背景切换上。本来可能的 4000 个单位的智力产出，减少到 2400 个单位。参阅 Gerald M. Weinberg 所著的 *Quality Software Management, Vol.1: Systems Thinking*（New York：Dorset House Publishing，1992）（中文版为《质量·软件·管理：系统思维》（第 1 卷），清华大学出版社 2004 年版。——编者注）。

模式 57　“棒球不相信眼泪！”



组织文化不鼓励人们表露情绪，进而使得冲突只能暗中进行。

> 曾经有一个从事广告代理的朋友告诉我：“在软件开发公司工作一定非常美妙，人们从来不会对其他人发脾气。”
>
> ——TDM

在外界看来，软件行业一定像是没有情绪波澜的避风港；在内部看来，则是完全两样。人们往往都慷慨激昂，而且在某些没必要的事情上也充沛着大量的激动情绪。从这个角度上看，软件开发行业与很多其他类型的知识型工作非常相似。

在那些过去几乎纯粹是加工型的公司里面，大多数的知识型工作仍然相对较新。以 AT&T 为例，几十年前，它的大部分员工还是低技能的蓝领工人，今天，里面全部都是知识型工作者。鉴于这样的经历，“工作场所禁止表露情绪”的潜规则在早期很多知识型工作的公司里面得以施行也就顺理成章了。人们常常会违反这条规则，可是它仍然延续了下来。在会议上哭泣或者针对不受欢迎的决定爆发怒火的人被视作没有职业态度，甚至无用。这样的人也通常会被视作在情绪上过于反复无常，因而得不到提升。最后就使得情绪和晋升相互排斥——这可不是成

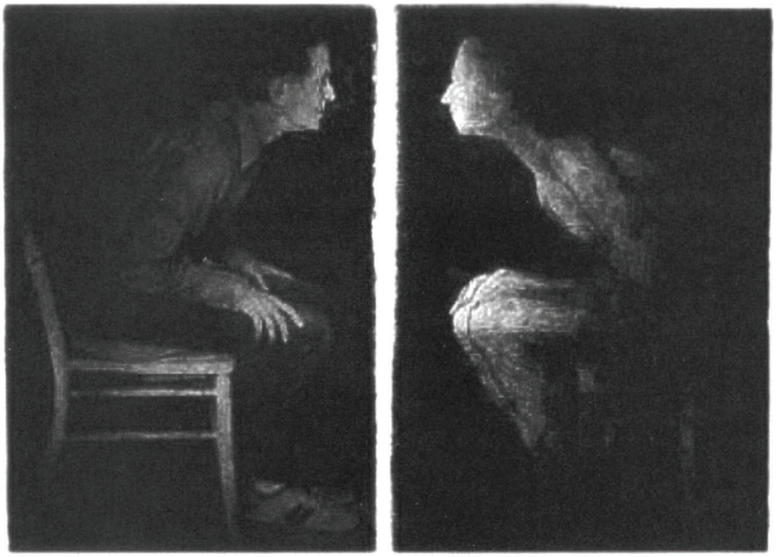
功之道。

在决定是否容忍难以控制的情绪时，请记住情绪对工作的干扰程度取决于人们对自己工作的关心程度。不让员工有任何情绪的简单办法就是招聘那些对工作不以为意的人。

给项目配备激情四溢、关心自己所从事工作的人员才是成功之道。他们的激情有时会掀起怒火，但是扑灭这类怒火是达成宏伟目标必须偿付的代价之一。

> 在 1992 年的电影《红粉联盟》(*A League of Their Own*) 之中，汤姆·汉克斯扮演了一位嗜酒如命的女子职业棒球队经理。他当着所有团队成员的面训斥某位队员，这位队员嚎啕大哭。他朝着她咆哮：“你在哭泣吗？不准哭。棒球不相信眼泪，棒球不相信眼泪！”

模式 58 铁窗喋血



“Confrontation 2”，Chaim Koppelman 作品，使用得到授权

合理的冲突被解释成“沟通失败”。

正如假设知识型工作不能包含任何情绪是一种错，把问题归咎于同事的打扰也是一种错。我们一直在针对问题错误地寻找原因，最常见的形式是把所有未能达成和谐的情景都归咎于沟通不足。沟通成了最终的替罪羔羊，即使最不善于自我批评的组织也乐意于批评自己的沟通技巧多么糟糕。作为咨询顾问，我们经常发现客户在我们前面作出这种自我批评，而具有讽刺意义的是，他们正是以最得体和最清晰易懂的方式自我批评。人们在宣称自己沟通技巧多么糟糕的时候，或许正是他们最善于表达自己想法的时候。

在保罗·纽曼的电影《铁窗喋血》（*Cool Hand Luke*）中，囚犯面对着一名以施虐为乐的监狱长，监狱长在每次体罚之前都会宣称“我们之间的沟通失败了”。如同很多其他的情形一样，这种情形中有失败，但却绝不缺乏沟通。在电影里面，双方之间的交锋是囚犯不屈不挠的反抗意志和监狱长粗鲁的、极其强烈的憎恶之间的交锋。每一方都十分了解对方。

下一次当你在组织里面听到有人把失败归咎于沟通时，请注意倾听弦外之音。那很有可能是“我非常了解你，但是我讨厌你说的话”。把它称为沟通失败，就把所有人的注意力从问题的真正根源——合理的冲突——转移开来，反倒集中到错误的根源上面。结果就是浪费了更多的精力去沟通早已被断然拒绝的信息。

其中深层次的原因是认为在业务背景下面发生冲突会显得不够职业。推论的过程如下：“我们都在为相同的组织工作，所以怎么可能存在冲突呢？除非有的人行为不够职业。”这个问题当且仅当你信任组织中的每一方都是各司其职时才有意义。组织是大型的、混乱的有机体，自然演化与刻意设计对组织成形的作用不相上下。组织之中的冲突无处不在。章程（如果曾经有人煞费苦心地去详加规定）里面涉及的某些价值观有时甚至互相矛盾。组织可能宣称决定把质量和生产率都最大化，但是这两者之间往往存在着折中。结合成一个整体的各个子组织有时冲着不同的方向使力：工程部门和市场部门总是不和，销售部门和财务部门的步调可能稍稍不一致，人力部门和企宣部门可能话语不和谐。组织中有些人的眼光可以看到 1~5 年之后，而另外一些人的眼光则可能只能向前看 1 天或者 1 个月。组织中普遍存在着合理冲突。

当冲突被视为非常自然而且绝对职业时，各方注意力就转向有效解决冲突的技巧上，而不是大错特错地去改进沟通。这样的结果也许不甚完美，但却总是更佳。

模式 59　按期交付，每回都不例外



团队总是按期交付项目版本。

时不时地，你能听到软件经理自吹自擂："我的团队总是按期交付。"这个论断真是让人印象深刻。假设团队已经多次交付，而且他们构建的软件也还比较重要，那么，如果总是（百分百地）能准时地在作为基准的计划日期交付，那实在是一个了不起的成就。

然而，总能按期交付的团队迟早会被迫降低质量门槛以赶上交付日期。当然，我们并不是说他们每次版本发布都是如此。但是如果一个团队对交付质量标准从不妥协，他们最后往往会错过交付日期。

在整个开发周期中，需要一直对优先级重新进行权衡并对资源重新进行分配。一般来说，组织运作项目有以下五种主要的"杠杆"手段。

(1) 人：安排谁上项目？

(2) 技术：团队能够使用什么流程、方法和工具？

(3) 范围：团队需要构建哪些特性？项目需要哪些平台支持？

(4) 日程表时间：团队打算何时交付？

(5) 交付质量标准：在交付之前，产品的完整性、正确性以及健壮性必须达到

什么程度？

如果制订了合理的计划，项目外部的这五个因素之间就取得了暂时的平衡。然而，随着项目的进行，一些因素发生了变更，而其他的因素也不再像此前想的那样。因此，你需要针对上述五个因素调整它们的组合关系，以使项目依然能行进在通往成功的康庄大道上。

我们已经在第 28 项模式里见到过，你离项目交付日期越近，上述因素中某些因素的效果就变得越差。比如以下情形。

- “往已经延误的软件项目中添加人手，会使项目更加延误。”[①]添加人手，既消耗了项目的日程时间，又消耗了项目上成员的精力。在项目活动的后期添加人手对按时交付很少有帮助。如果一定要说它会给项目带来什么，或许就是延误吧。
- 更改开发方法或者工具需要重新培训，而且对于第一个尝试新流程或者工具的项目而言，也不大可能加快开发的速度。学习曲线也消耗了时间。
- 缩减范围只有在被消减的特性还没有开始开发时才能发挥作用。版本的整个周期里面常常会出现这样的时刻——产品特性已经基本完成，大多数的特性已经完成编码，团队正在使其稳定下来。你可能会砍掉一个已经完成的特性以节省 QA 的时间，但在整个周期的后期缩减范围其实已经没多大用处了。

当版本周期的后期出现了问题时，你往往会发现自己只有两个杠杆可以调节：日程表和交付质量标准。如果处理得当，你在项目后期发现的问题还不至于酿成重大的问题，但是纠正它们仍然需要一个过程。

如果你确实承诺了严格地按时交付，并且每次都不例外，你剩下唯一可行的纠正手段就是：放宽你的交付质量标准。

---

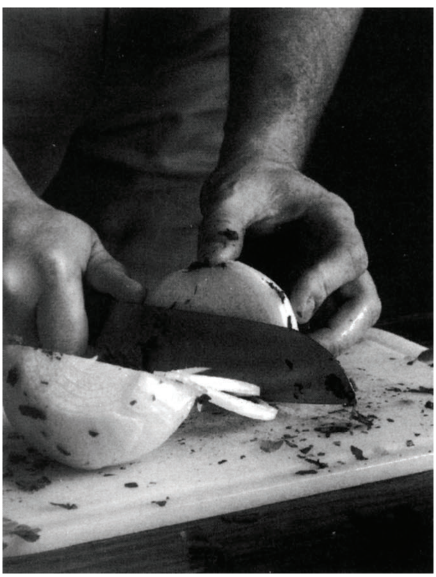
① Frederick P. Brooks 所著《人月神话》（*The Mythical Man-Month: Essays on Software Engineering*）（Reading, Mass：Addison-Wesley，1975），第 25 页。
（该书英文版已由人民邮电出版社出版。——编者注）

模式 60　食物++

项目团队成员定期在一起享用他们的食物，而且如果可能，整个团队会在一起筹划和准备这些食物。

在奇幻的动漫电影《千与千寻》的制作过程中，导演意识到除非动画团队加快他们的速度，否则电影将错过夏季首映。团队认为摆在面前的最好办法就是延长工作时间。

有一天晚上，所有人都工作到很晚，团队中的一位画师承担了烹制茄汁熏肉意大利面（*spaghetti all'amatriciana*）的任务。仍然在加班的每个人——此时还有相当多的人——一起享用了晚餐，并且宣称他们喜欢这种经历。在第二天晚上，另外一个团队成员决定他来给所有人烹饪，然后在下一个晚上，其他人又接着烹调。于是，一项团队的传统就形成了。



在每个晚上，团队里面的某个人会为大家准备一份晚餐。就连导演宫崎骏也决定展示一下他的厨艺，做了一份精心烹饪的面条。为团队的其他人下厨这种简单的做法有着激励的效果：动画团队赶上了截止日期，电影按时上映。

围绕着食物的固定程序——准备、食用时的协作、清理过程——所有的参与者之间建立起了一种联系。

我们曾听说过一个对食物非常着迷的团队。午饭时刻，这个团队的一个或者多个成员会去重新布置自助餐厅的桌子——这绝对违背了自助餐厅的政策——好让整个团队的 16 个人可以坐在一起。有人占座位来阻止擅入的人，直到整个团队就坐，一起享用午餐。

这些团队成员不是因为项目经理的要求才坐在一起吃——他们是觉得这属于团队建设的一部分，他们希望大家在一起吃饭。

如果晚上需要加班以赶上某些急迫的截止日期，那些没有紧急工作需要处理的团队成员就会驱车前往超市买回一些食物。他们原本可以回家吃饭的，但他们选择留下来为晚走的同事提供食物，并且一起进餐。

当一个团队筹划或者准备食物的时候，你会发现奇妙的现象。首先是收集原材料的过程。这毕竟不是快餐，一些团队故意选些难找的原材料，使得寻找的过程变得困难，从而让团队变得热闹起来。然后下一步是准备：技艺娴熟的厨师负责困难的准备工作，厨房奴隶（他们被亲切地给予了这个称号）负责鸡毛蒜皮的小事，还有其他人布置餐桌或做其他的工作。

当食物端上餐桌的时候，它是整个团队一起努力的结晶。“我们做出了这个，我们所有人一起做出来的，现在就让我们一起享用吧。”这是整个团队的共同想法，所有人坐下吃共同努力而来的食物。对于长期在杂乱的项目上一起工作的团队，这是“项目中的项目”，可以快速地完成和品尝。

你同样可以在世界各地看到人们因为食物而集聚到小餐馆的例子。人们定期在咖啡馆召开商业会议。笔记本电脑、纸张与卡布奇诺[①]、羊角面包一起竞争着桌上的空间。分享食物——我们把咖啡也视为一种食物——带来的亲密感使会议变得更有成效。那些想和客户建立牢固关系的销售人员会频繁地光顾餐馆，这尤其是一个有力的佐证。

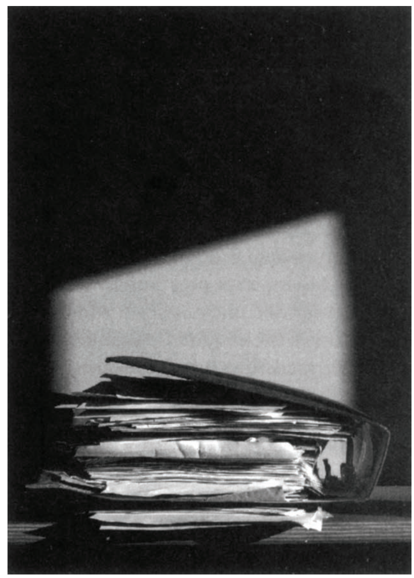
一起吃饭并不能保证你的团队就会成功，就像不在一起吃饭也无法得出项目必然失败的结论一样。然而，我们观察到很多成功的团队喜欢一同准备和食用食物，他们恰恰利用了这一过程中丰富的互动机会。

① Cappuccino，一种咖啡，由蒸汽加压所煮，并加热奶。——编者注

模式 61 没人在意的交付物

没有人愿意为团队开发的一些项目产物掏腰包。

系统与软件行业的每一次流程改进都定义了连篇累牍的新活动、角色和产物。比如 Rational 统一模型和德国的 V 模型，每一种模型都建议了 150 多种的交付物。这些交付物包括需求规格书、设计文档、详细模型、用户界面概念设计、测试计划、项目估算……这个列表似乎无穷无尽。



James Rye 摄影作品

这些交付物之中有一种从未被质疑，那就是最终的交付物：产品本身。可是，其他的交付物都有什么价值？我们是否需要所有那些交付物？是否值得投入时间和精力来生产它们？

有时，团队花费时间来开发没人在意的产物。人们也意识到在时间和精力上有所浪费，但是流程要求团队必须产出每一种交付物。在这样的情形下，出现了一个问题：到底谁是这些产物的资助人？

资助人：预先承诺偿付金钱以支持某项活动的人或者组织。

每一项产物都需要对应有愿意付钱的资助人。在这种场景中，付钱意味着不仅仅有权利要求开发某项产物，而且也有能力把必要的开发资源拨派给团队。

如果只是项目内部要求上述问题中的交付物，那么做这个决定就很容易了。直接由项目经理决定开发该产物是否帮助或者妨碍了项目目标的达成即可。

如果项目团队以外的某个人需要或者想要某些交付物，而开发那些交付物又被项目团队认为是一项额外的负担，这种情形就不是那么简单了。在这种情形下，项目必须要确认该交付物的资助人。

想想如果组织希望所有的项目都提交标准的软件架构文档，会发生什么。其目的是鼓励设计方案的重用，并且让项目外部人士清楚项目的情况。所要求的架构文档引人入胜地概述了最重要的决策，并且按照预定义的格式展示了项目的概况。然而，开发团队的关键目标是在预算内按时完成交付。他们根本不需要这份文档就能达成他们的目标。中央架构组在这种情况下可能会担任起资助人的角色：该架构组可能会在项目启动之初派出一名组员帮助团队遵循公司的架构标准，并且帮助团队编写出这些要求的文档。

除了标准用户手册之外，开发客户专用的用户文档可能需要有一位营销部门的资助人。为了这份文档能够及时完成，他也许会派遣一名营销人员加入团队，负责针对当前的版本编撰这些特定的用户指南。

另一位资助人可能是中央人机界面组的组长，他希望能开发一些界面模型以吸引用户早期的反馈，而且在这项工作上有一笔劳动力预算。

项目之外的其他人也能扮演资助人的角色，比如中央质量组也有预算来收集关于版本发布之后故障排除率的长期统计数字。这位资助人向项目提供了人力或者时间和金钱来处理这些额外的工作——项目经理往往很不情愿分割自己的预算来做这些工作。

在不需要负担支付费用的责任时，就很容易轻易要求开发某些东西。大多数的中央方法和工具部门都不向项目提供资源。他们的工作就是提议那些他们觉得不错的主张，游说每个项目与之保持一致。但因为他们都是“有职无责部”的部长，所以他们的提议未必能受到项目的欢迎。

没人在意的交付物并没有清晰一致、经过验证的需求，换句话说，没有资助人。考虑一下每一个没人在意的交付物的价值。如果你发现没有人愿意为之付钱，而且项目也不需要它，就不要去开发。假如你认为它是个不错的想法，但却没人会为之付钱，那就找到一个资助人。

模式 62　隐藏的美



"Copenhagen Underground"，Michael Altschul 作品，使用得到授权

项目的某些产品不是满足于尚可甚至优雅的标准……而是追求至善至美。

我们有些人的工作就是为了吸引眼球。比如你给汽车设计一款新的车体造型，那么最后的成功在很大程度上是取决于其他人对之有多满意。如果他们对所看到的感到满意，你就能从他们的反馈中感知到这一点，并体会到快乐和尊重。如果你足够优秀，这份快乐就相当于你大部分的薪酬。剥夺你的这份快乐就如同不给你偿付报酬一样，都违反了你的雇佣协议。

现在想象一下，你设计的不再是车体造型，而是汽车安全气囊的自检系统。几乎没有人会留意你的工作结果，甚至他们也只是大概知道有这么回事而已。所以，这项工作的成败以及可能随之而来的满意度，就只取决于产品是否达到了预订标准，完全不关乎美学要求。

多么荒谬！设计本是一项从无到有的创作过程。创作的思路可以把你引向诸多不同的结果，也许那些设计在功能上是一样的，但在我们称为"美学"的方面肯定存在差异。有些设计就是很美。它们的美，不是附加物，不是"修饰"，而是在——以自然却不失惊艳的方式——完成功能的过程中出现的副产物。对于那些

可视的设计是这样，对于那些大部分不可视或者整体不可视的设计也是这样。

> 因为以太网的发明者 Bob Metcalfe 是我的一位朋友，我觉得也许可以看看以太网协议的细节部分以了解它的设计思路。我打开了即将公布的规范，虽然尚未添加修饰，但出乎我意料的是，我发现协议充满了美感。详略得当、概念优雅，而且对包丢失的恢复机制跟包原始的传输方式如出一辙。协议里冲突的概念和处理是我所未曾料想到的，但至少在我看来，它非常地简洁。你们也许会笑我小题大做，但这个规范的好处的确让我难以忘怀。
>
> ——TDM

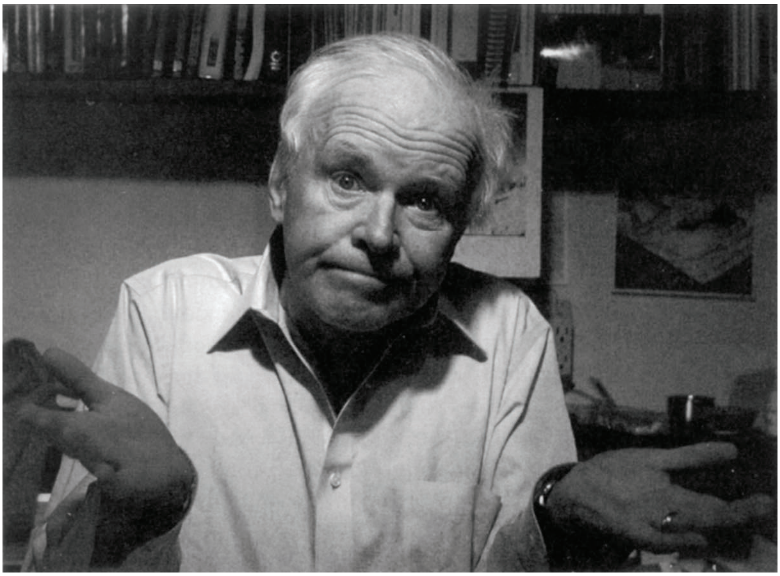
所有的设计都存在美学元素。问题是，这种美学元素对你是敌是友？如果你是一位经理，特别是阅历尚浅的经理，你也许会担心设计人员作品中的美学元素可能是一种浪费，不过是我们被反复叮嘱应当避免的镀金行为。经理对美学的这种无动于衷的态度使设计人员看不到什么是卓越的工作，拒绝了那些“尚可”标准之上的更高标准。

与之相反的态度则要求你有能力、也有意愿去研究手下人的设计细节，并且洞察其中蕴涵的品质。即便只花很短的时间来这样做，你也会马上意识到所谓的镀金讨论不过是为了转移注意力罢了，任何设计都不可能通过堆叠附加特性或者外在装饰得到改善。恰恰相反，只有*减少*特性才能改善设计的美学品质。最好的设计都是简洁的、功能明确的、易于测试的，而且即使对其修改，也不会带来新的麻烦。此外，它们让你觉得没有比这更好的方式来实现产品的功能了。

在工作成果很大程度上不可见的情况下，设计人员会受到关注细节的经理的极大影响。如果你用心去欣赏某位设计人员的作品，可能就会进入另一个世界，从而发现以前未曾注意到的美妙细节，两个人因此也就有了默契。在设计人员的眼里，你也许就会从一个还不错的经理转变成“我会一直跟随的老板”。

> “美到极致不是增无可增，而是减无可减。”
>
> ——Antoine de Saint-Exupéry

模式 63　我不知道



组织营造出能讲真话的氛围，即使讲真话意味着无法立即给予答复。

你正出席一个会议，项目经理问你当前的数据库是否包含了计划的市场分析所需的全部数据。你对那个数据库及其里面的内容相当熟悉，而且你知道数据库里面包含了大部分的必需数据，但是你对某些内容并不肯定，而那些内容对于市场分析是至关重要的。

在这种情形下，对项目经理如实的回答是“我不知道”，但是在你的组织里面说这句话是否安全？又或者你是否被迫掩盖、逃避、竭尽全力来避免说出“我不知道”？另一方面，如果你坦承自己不知道，你的同事听到了会不会觉得你不够优秀？

在有些组织里面，“我不知道”被理解成“我现在没有一个答案，但我最终会给出答案”。这句话为后续的讨论以及探索找到答案的途径打开了一扇门。

“如果你说真话，你就没有必要记住任何事情。”

——马克·吐温

假设你作出如下回答：“我现在还不知道，但是如果我能与乔万尼——他曾参与了原始的设计——一起花上几个小时的时间，我相信我们能给出那个答案。”在这个时候，阿戈斯塔插话进来，说：“嗨，我在另一个项目上对那些数据做了一点分析，而且做了一些笔记。我觉得它们也许能回答你的问题，它们或许可以帮助你节省大量的时间。”除了给出实事求是的回答，“我不知道”也能激起团队的协作氛围，鼓励每一个对问题有一定了解的人提供有帮助的信息。我们谈论的不是持续的无知状态，而是公开宣布的待填补的知识缺口。

当绝对不容许说“我不知道”的时候，就会出现与本模式截然相反的情况。人们既不知道答案，又没有足够的安全感来坦承这一点。或许是他们害怕失去同事的尊敬，或许是他们害怕老板对自己产生无法做好本职工作的印象。此外，“我不知道”可能会被经理视为一种威胁：当他孤注一掷地抱住进度表时，他毫无疑问不希望出现未知因素，更不希望这种未知因素会威胁到项目进度。开发流程中存在着明显的未知因素的想法绝对不能被接受。大多数此类组织把开发流程看成是工厂中的流水作业线。在流水线上几乎不存在未知因素：“当汽车到你的跟前，拧上相同颜色的后视镜。然后等待下一辆车。”如果开发团队成员说“我不知道”，就相当于说“停止流水作业，等我弄清楚后视镜朝哪个方向”。这一类组织宁愿保持着流水线运转的状态（即便缺少必要的组件），也不愿生产过程出现短暂的停顿。

“我不知道”在有些组织中被视为说话者的怯懦标记。这类组织的文化是每个人都应该无所不知，尽管我们都非常明白自己并不是“百事通”，而且从来都不是。这种闭目塞听的态度只会导致人们不愿意去寻求帮助——即使他们很明显需要帮助。结果是事情拖得更久。但是，如果开发中的项目错过了截止时间，组织宁愿把其他的因素拿出来当作理由——无能的管理层、缺乏人手等等几乎一切理由——也不愿承认自己的开发流程可能包括未知因素。

每当有人说“我不知道”，你听到的就是信任的宣言。如果整个组织里面的人都觉得说出“我不知道”是安全的，这说明人们觉得寻求帮助是安全的。这一类组织真正是在各个层面上都鼓励协作，并获得由此而来的好处。

模式 64　乌比冈湖儿童

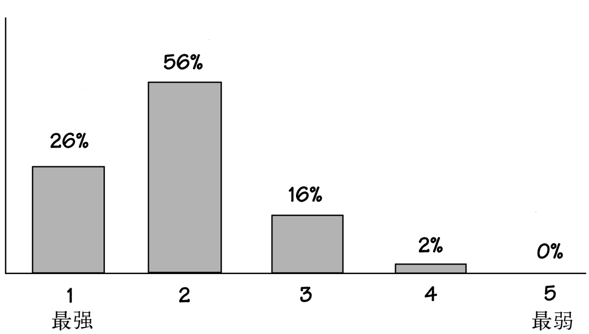


Marja Flick-Buijs 摄影作品

经理给出的绩效排名不能有效地区分出执行力的强弱。

在电影《牧场之家好作伴》（*A Prairie Home Companion*）中，公共电台主持人加里森·凯勒播报来自于虚拟小镇乌比冈湖的新闻：“在那个地方，所有的女人都很强壮，所有的男人都很帅气，所有的儿童都智力超常。”

绩效排名经常会像乌比冈湖儿童的智力等级一样，只分布于一个狭小范围之中。想想下图所示的例子。



这种模式表明管理层在应对末尾绩效和辨别超常绩效方面陷于失败。“乌比冈

湖效应”是破坏性的，最主要的原因是它揭示了撒谎的文化。

根据上图显示的绩效分布，你可以很有把握地指出一点：这绝对不是事实。多少年来，我们是否已经看到了一贯如此的报告——无论何种规模的大型团队，就个人而言，团队成员在软件工程中的绩效表现都变化巨大？我们认同这些报告结果，部分原因是因为它们符合了我们自己在工作场合观察得出的结论。可是，在多得数不清的公司里面，每年一次或者两次提交给 HR 的绩效评定往往表现出大家不相上下的态势，几乎人人都是“高于平均”的平均水平。

“乌比冈湖效应”通常源于诸多因素：HR 所产生的混乱、高级经理的愚蠢以及团队领导的怯懦。

当人力资源的专业人士针对绩效的评估与级别提出了互相矛盾的指导方针时，他们就制造了混乱。一个常见的例子是在相同的绩效评定系统中混合了绝对和相对的标准。我们看见过很多这样的系统，在有些方面以绝对含义的词汇描述职位要求，从而定义不同级别的绩效（例如，1=超出职位要求，2=与职位要求一致，3=大抵满足职位要求，4=没有满足某些职位要求等）。在同一个系统的其他地方，却明文规定评定结果理应符合既定的级别分布——也就是说，百分之十到十五会是“1”，百分之二十到三十会是“2”，百分之四十五到五十五会是“3”等。这些百分数暗示了评定结果是相对于员工的总数来说的，而不是某些绝对的特征。

高级经理有时也会发明一些激励措施来避免出现那些绩效不合格的雇员。经典手段之一是下面的“左右组合拳”。

(1)“我们正在建设一个高效的学习型组织，因此所有绩效评定为‘4’或者‘5’的雇员必须接受绩效改善培训，否则会被‘设法弄出’组织。”

(2)“由于短期的预算压缩，所有公开的（新员工）招聘和置换请求已经被临时冻结，直到后续通知。”

在这样的情形下，不止一位经理会得出这样的结论（有时是错误的）：从绩效糟糕的瓦尔多那里得到的净工作量一定会多于从一个空椅子中得到的。

跟把事情责怪到 HR 和愚蠢的执行官头上一样有趣的是，经理们不去面对末尾绩效的最常见原因在于自身：很难做好绩效评定，总之，与一位绩效糟糕的人真正进行一次坦诚的讨论总会让人感觉不是滋味。于是，经理们放弃了。

有些时候，为了与绩效糟糕的雇员进行有建设性的交流，经理得稍微变换一下角色。在绩效管理的早期阶段，经理处于教练的模式：解释、演示、协助、回

答问题，尤其是要鼓励该雇员。等到了评估绩效和提供评定结果的时候，经理必须更多地扮演法官的角色。“这是你所完成的，”他说，“这些事情做得不错，其他的事情仍有改进的余地。”诸如此类。

当雇员没有达到其角色期望值时，他通常意识不到自己的不足，直到并且除非经理把自身的角色从教练切换到法官，他才会认识到不足。因为这种角色切换相对而言不常发生（一年之内很少会多于四次），而且几乎总是令经理和雇员都不舒服，所以来自法官角色的声音很容易流于沉默，甚至被忽略。

除了乌比冈湖效应之外，未能应对末尾绩效的另一个外部症状就是“缩水的工作”。瓦尔多不及格——而作为他的经理，你在应对他的糟糕绩效方面也不及格——你也许反而会从他的手头拿去某些事情（那些需要正确完成的事情）。他那些更为胜任的同事（和你）拾起他落下的活。慢慢地，但也用不了很久，他表面上的绩效就会逐渐改进到可以容忍的级别。当然，这只是因为他现在做的事情远远少于其角色所要求的事情。

只使用绩效评定范围的狭窄区域、“缩水”一些工作有什么错呢？答案很简单，那就是这对于其他的团队成员不公平。在他们处于什么水平这个问题上，你跟他们撒谎了，因而也就剥夺了他们管理个人职业规划所需要的信息。

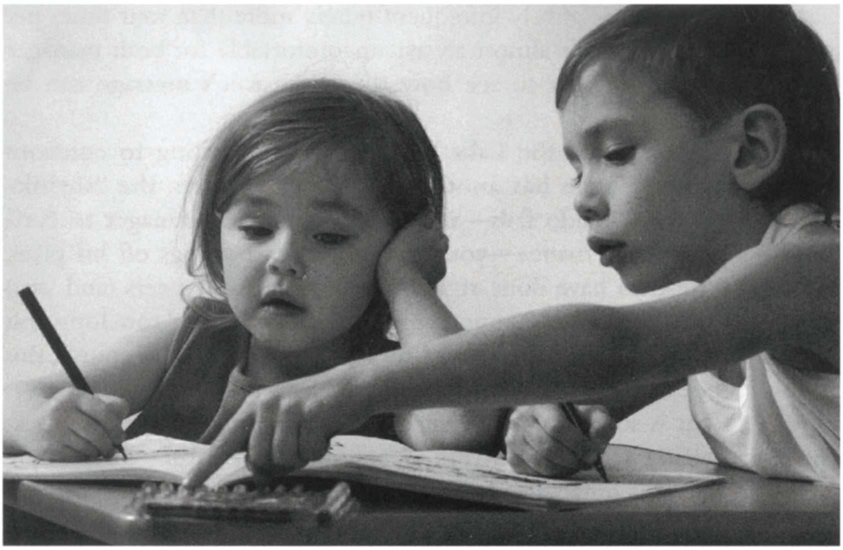
你对那些绩效卓越的雇员撒谎了，因为你没有让他们知道他们的工作多么出色（和令人赞赏）。让我们面对事实吧：每个团队成员都知道哪些人的表现出类拔萃，包括那些表现出色的人自己也能知道。表扬作出那些贡献的人吧，那是我们应该做的。如果那意味着需要给出极其卓越的绩效评定，那就给吧。

你对那些绩效较差的雇员撒谎了，因为你没有及早警告他们的绩效很差，而这已经关乎他们能否继续从事这个职业。有时，糟糕的绩效源于未能理解这个工作提出的期望值；有时，雇员能够而且的确会改进他们的绩效。如果他们能够及早地意识到自己没有达到期望，接下来他们或许能够成功达到甚至超出期望。

你对那些评定处在中间位置的人撒谎了，因为你所表明的处于中间位置的人数要大于实际情况。绩效优秀却没有得到最高评级的员工没有获得应有的赏识和奖励。与此同时，如果有一半的雇员都得到了“3”分（5 分系统），那么相对于真实情况而言（本该只有百分之二十的人得到“3”分），绩效属于末尾却得到一个“3”分的雇员会觉得更为惬意。

只有基于实际的平均水平，你才讲出了真正的事实。

模式 65　互相教学



Dmitriy Shironosov/iStockPhoto 版权所有

项目的利害相干人明白每个人都能从其他人那里学到很多东西。

在组建项目的时候，目标就是更改某些系统、某部分工作、产品或者服务的状态。这一点是很明显的。

然而，一旦项目开始，期望每个人或者每群人都准确地知道理想的未来状态会是什么样子，这是不现实的。

例如，在项目的起始时刻，构建者期望用户完整、精准地列举所有的需求，这是不现实的。类似地，消费者也几乎不能期望构建者在分析完条件之前，就可以理解需求。在不理解需求的情况下，改变的努力几乎总是折戟沉沙。

问题通常在于利害相干人不明白他们必须向其他人学习。在消费者与构建者之间必须互相教学。每一方都必须教导对方，并向对方学习。

让我们更具体一些来看。需求收集者需要研究消费者的工作，才能确切地指出有效的产品或者服务。然而，有三个障碍也许会阻碍消费者把他们的知识传递给构建者。首先，消费者可能太熟稔于他们的工作，以为一切都是不言而喻的——他们未能提到细节（他们以为构建者已经知道了）。其次，消费者并不一定具备良

好的沟通技巧，可能对主动提供有助于构建者的信息变得不耐烦。最后，消费者可能在确认或预料他们未来的需求方面存在困难。在我们看到实物之前就清楚我们想要什么东西，这是非常困难的（甚至是不可能的）。

想一想漫步于书店的经历，你可能会突然发现自己对某一主题的某本书很感兴趣，而你先前却从未考虑过这个主题。或许你并不经常浏览书丛，那么想象一下在你最喜欢的购物类别里面发现一款令人激动的商品——也许是一件时髦的电子器件或者梦幻般的女士内衣。（读者请基于自己的性别进行选择。）简单地说，在知道某些东西是否符合自己的需要和目标之前，我们很可能要先观看或者体验一下。

所以，如果构建者必须向消费者交付如同书籍、电子器件或者女士内衣一样令人愉悦的产品或者服务——而预先又不知道产品或者服务的特征，就必须有双向教学的过程。不幸的是，有一些障碍往往会阻碍消费者和构建者进行有意义的学习：已签署的规格书、误导性的解决方案、关闭的学习之门以及通用语言的缺失。我们将在后面讨论这些。

有些组织坚持在任何开发活动开始之前都要签署需求规格书。规格书通常由主要的消费者签署。由于主要的消费者将对规格书全权负责，他自然而然会去强行指定。这使得需求收集者的角色退化成抄录员，阻碍了本应基于早期原型和探索性模型进行发现与学习的过程。

互相教学也可能会因为误导性的解决方案而受到阻碍。有些利害相干人把自己摆在知自己之所欲、而且该欲求终归会满足的位置上面。在这样的情形下，他们想要的就是对于当前问题的直接的解决方案。但是，那些解决方案很少能够解决大量更大的需求——那些关乎工作条件、承担工作的组织、组织之外利害相干人的需求。在这样的情形下，每个人都需要理解问题本身，做到了这一点，解决方案迟早会出现的。

学习最好是及早开始。随着项目进展，想法开始定型，期望更加不容易改变，曾经谈论的解决方案越来越富于预见性。对于构建者和消费者，打破这一切并就他们想要的产品或者服务学习新的东西就变得越来越困难。一旦错过早期的机会，需求提供者和需求收集者的传统角色就变得更加根深蒂固，创新就变得难上加难。

每一位利害相干人都知道一些事情——往往知道的还不少——而且通常他们所了解的知识还各不相同，所以他们必须互相请教。然而，要想使背景不同的各

方有效地互相教学，他们需要一种通用语言——“世界语”项目（如果你愿意）。通常，建模语言的效果最好。互相教学必然会需要反复试验，而建模是理想的手段。当人们知道每个人都只是把讨论中的模型视为学习过程中的一次性成品时，他们很少会去为自己钟爱的特性大声争吵，而更可能去思考学到的教训。

如果要让开发项目产生正确的产品和服务，就必须深刻地理解消费者的需求和支撑那些需求的特性。这种理解只有当消费者和开发者各自向对方学习需求的时候才会出现。最难做到的就是意识到同心协力互相教学的必要性。

模式 66　意气相投



组织允许特殊的团队来简化它们开发流程中的规则，甚至是那些最基本的规则。

下面是你在这种团队中第一眼就能察觉的一些特征。

- 他们嘲讽预定的会议，但是自己却召开大量小型、临时的会议，几乎所有这些会议都是——或者马上就会成为——设计研讨会。
- 相对于使用其他媒介，他们更愿意使用白板来记录下想法、设计以及待办事项。
- 他们从尚未彻底完工的、高层次的需求陈述开始工作。他们经常跳过书面的设计文档，在开发的过程中早早地进入编码阶段。
- 他们扔掉大量的代码，然后进行重写。一旦特性经过了演示，他们就开始对其返工。
- 他们非常、非常快速地完成所有这些事情。特性的开发时间通常是 1 到 3 天，极端复杂的特性可能会花上 10 天。很多任务都在不到半天的时间里面

完成，并准备好测试。

这不是一个普遍的模式，但它是如此地不同寻常，值得人们注意。由于缺乏更恰当的称谓，我们把这些人称为“游击队”。他们在敏捷社区中更为普遍，[①]但也有一些团队独立于具体的方法而产生出了“游击”行为。

在有着传统软件开发背景的人看来，“游击队”的某些做法颇令人不安。他们看上去鲁莽大意，但无可置疑的是他们以惊人的速度取得了真正的进展。他们的做法会让你重新调整自己以往的那些价值判断，从而让你觉得软件开发流程中的很多部分其实没必要非得那么正式。

我们在软件上做的许多事情都基于一些基本的理念，比如“寻找和纠正一个问题的成本随着开发周期的进行而急剧上升”。因此，我们希望尽可能早地把需求（和设计）整理正确，避免后期的返工。这里有一个更为基础性的关于工作的假设：工作软件的构建和验证太昂贵了，根本无法多次重复执行，一个项目里面只能有一次或者两次这样的行动。虽然这一点在通常情况下也许是对的，但“游击队”能取得成功，是因为他们不属于这种通常情况。他们开发和测试代码的速度是如此之快，以至于他们在知道需要构建什么产品的时候，真正能够承担得起抛弃大量的代码所付出的代价。

那么，“游击队”对什么情形有效？嗯，肯定会对新产品的第一个版本有效，有时候也会对第二个版本有效。这些团队擅长探索较新问题域并为之设计创造性解决方案。虽然他们也能够生产出很优秀、很耐用的代码，但“游击队”汲汲于创新。

“游击队”就像没有安全设施的电动工具。他们可以有高得惊人的生产率，也可以是惊人地富于破坏性，这取决于如何引领和指导他们。他们也存在着一些内在的局限性。

“游击队”是个有机的结构。他们不可能很快地创建，也绝不可能通过命令就创建出来。他们通常是围绕着一两位引人瞩目的领袖人物形成的。他们缓慢地吸纳新队友（因为他们的标准非常之高），而且只有在领袖的领导下，他们才会团结在一起。他们可以很多年都保持高效，但一旦团队开始分裂，它转眼就会消弭于

---

① 你能在 Alistair Cockburn 的 *Agile Software Development: The Cooperative Game*（第二版）（Boston: Addison-Wesley，2006）一书中找到一项很好的针对于敏捷方法以及底层原则的调查。或者访问 www.agilealliance.org 作为入口点。

无形。

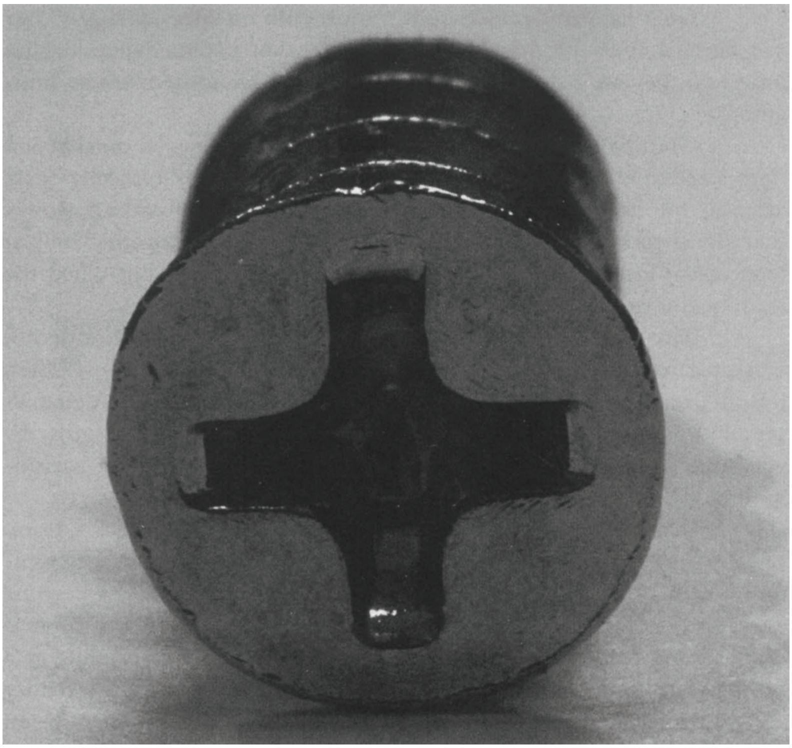
他们的规模不可避免地很小，并且所有人坐在一起。“游击队”与其他团队的合作也不是很好，尤其是远程的团队。他们团队内部的凝聚力相当之高。这个模式需要团队对外部的依赖非常松散才行。于是，他们不会成长得很大，他们不能被重新安置在其他地方，他们与外部的交往也不会太和谐。

知道何时停止使用“游击队”是非常重要的。因为这样一类开发人员在开辟新领域时会兴致勃勃，而一旦产品或者系统确定下来，他们通常就会失去兴趣。“游击队”在任何系统上都很少会坚持到第五个版本。曾几何时，你或许考虑过把团队转变成更为常规的结构。你就得为你的“游击队”寻找需要征服的新鲜领域，否则他们就将在其他某个地方为自己找到那些新鲜领域。

最后关于“游击队”的警告：小心大量存在的滥竽充数者。因为这个模式格外吸引处于各个能力层次的开发人员，所以很多小团队都认为他们自己是“游击队员”，但真正的“游击队员”非常稀少。在把你的项目押注于这样的团队之前，你需要确认自己是在跟切・格瓦拉[①]打交道，而不是切・雷诺这样的冒牌货。

① Che Guevara（1928—1967）出生于阿根廷的古巴游击队领导人。他于 1959 年参加了卡斯特罗领导的古巴“七・二六运动”，推翻亲美政权。于 1967 年被玻利维亚军队杀害。——编者注

模式 67　十字槽螺丝帽



惊人的是，显而易见的好想法不会很快被接受。

想象一下美国的发明家亨利·菲利普斯（Henry F. Phillips）是如何在 20 世纪 30 年代神奇般地发明了如今到处可见的十字槽螺丝和螺丝刀。他的发明无可辩驳地优于早先笨拙的开槽螺丝。在偶然见到这种老旧的开槽螺丝时，你也许会想起螺丝刀总是从开槽螺丝上滑下来，导致你中断手头的工作，不断地咒骂着。十字槽螺丝刀就不同了，因为它是自定位的，而且从来不会滑出。

这项新发明无疑更好，但气人的是，当时人们一直都在用开槽螺丝。菲利普斯一定是心情烦躁——他这项更好的发明就这么被忽视了。虽然他的发明最终被广为接受，但他并没有等到那一天。在今天看来，你也许希望能够返回到上世纪 30 年代，对他说出一份鼓励之词。

“坚持住，亨利，”你告诉他，“历史将会被你征服。到时候，你简直无法想象还有哪一款新产品会使用开槽螺丝——那一天会到来的。十字槽螺丝将处于统治地位。”

“是的，但要等到什么时候？什么时候？”

“嗯，这也许会花上几年，但那个时候——”

“几年！！！你是说可能在 1935 年或者 1940 年以后，人们才会接纳我的发明？”

“事实上，我们觉得更可能是在 1985 年到 1990 年之间。”

“啊！”

更新、更好并不足以保证立刻能被接受，那得一段时间。组织抗拒变化，或者在决策的延长期里面推迟变化。但是那些发明和支持更好想法的人，看到自己的建议被人忽略，或者更糟，直接被枪毙掉，往往会变得垂头丧气。在军事上，考虑死亡被称为“慢滚”。在项目工作的这么多年里，我们看到几乎每一个新出现的好想法都曾经是“慢滚”（至少在短时间内是这样的）。即使在像软件这种所谓高速发展的行业里面，举个例子，一些今天已经广为接受的最佳实践也花了差不多 20 年时间才被接受。[①]

如果你正因为没人采纳自己所支持的、明显更优的方式而垂头丧气，那么请从下面的事实中振作起来：我们以往的伟大发明家，像托马斯·艾尔瓦·爱迪生和韦尔纳·冯·西门子这样的人，都不是仅仅因为某项发明就被人铭记，而是因为他们一次又一次提出了新想法。这种能力才使得他们与众不同。能推动一个想法付诸实施的人是倡导者，而那些历来总能提出新想法的人则是发明家。由倡导者到发明家的转变需要很多年，甚至几十年，但这是踏踏实实、不断渐进的工作，并且还伴随着一项惊人的额外好处：人们更愿意去接受由经过考验的发明家所提出的想法。

① 对于落后 20 年的解释，参阅 Samuel T. Redwine 和 William E. Riddle 撰写的“Software Technology Maturation”，第八届软件工程国际会议的论文集（New York：IEEE Computer Society Press，1985），pp. 189-200。

模式 68　可预测的创新



Gabriel Bulla摄影作品

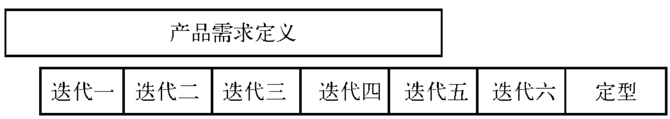
团队在自身对创新的需求和老板对可预测性的需求之间做出平衡。

如果你构建的所有系统都非常有趣，那么你应该知道创新是每个项目的一部分。创新意味着你的团队将会做一些从来没人做过的事情，而且该投入是否成功依赖于人们是否使用全新的、不同的方法来解决问题。与此同时，老板几乎总是要求你就“什么时候完成”这样的问题，提供比较精准的预测。这等于是让你去预测好的想法何时出现——那可不容易。

你也许非常明白这种困境：就像是走钢丝绳——既要给你的开发人员提供足够的时间来探索、发现、了解和解决问题，又要给你的老板和客户就“开发工作何时完成”提供精确的预测。走钢丝绳很容易偏离重心，倒向其中一边。如果设定的期望值过于乐观，你一定会给你的开发人员带来很大的压力，从而限制了他们开发正确产品的能力。另一方面，如果你一直拖到开发周期的后期阶段才去预测，你也许会发现老板对你个人的绩效不甚满意。

开发计划的流程是像这样的：即使产品特性需求的数目适当，你也应该明白你的开发人员在弄清楚要构建什么以及如何去构建之前，可能需要一些迭代时间。因此你在早期计划了两到三个固定时间的迭代，来开发原型和进行探索。每个迭代持续多长时间取决于任务本身的复杂度和大小，但这些迭代通常每个都会持续1到4周。

在两三个这样的迭代之后，大多数技术上的未知因素都变成了已知因素，最终的产品特性可以进入开发了。这以后的迭代同样重要。团队通常再利用三到五个迭代的时间完成版本开发。再一次强调，每个迭代的长度因团队不同而存在差别，但是它们的长度往往在4到12周之间。所以，版本整体的开发时间线如下所示。



在这个例子里面，前三个迭代集中用来对整个特性域进行探索和原型制作。这并不是意味着迭代一就要把交付产品的所有特性原型都开发出来。它只是意味着开发人员可以在产品的任意方面（由他们自己选择）自由地发挥。它还意味着迭代一的结果在迭代二开始之时也许会被完全抛弃。虽然迭代一的某些代码很可能成为迭代二工作的基础，但那并非必然。迭代一（和迭代二，如果有必要，还有迭代三）的目标是澄清未知因素、减少开发工作中存在的不确定性。总而言之，迭代一到迭代三可能会花上6到12周的时间。对于这个时间长度，每个团队有可能会不一样。

最后的那些迭代（根据具体的情形，可能是三个、两个或者六个迭代）要把最终的交付产品构建完毕，其中极有可能会用到起始三个迭代开发出来的部分代码。这些迭代花的时间往往比探索性迭代的时间要长，它们的长度主要取决于团队使用的开发方法，通常长达4到12周。

这种方式如何帮助平衡创造性和预测性？要想取得平衡，就得在固定的开发周期里面，多次地对问题域中需要创新的那些领域进行探索。那这能否确保你的团队在合适的时间就一定会得出需要的好想法？当然不会。但这种方式可以让我们在整个开发周期内规划一些创造性的迭代。

项目因为创新而不断前进，但它们同样也需要可预测性。但是，两者中哪个太多或者太少，都可能把团队推入深渊。

模式 69　玛莉莲·明斯特



在有些组织中，开发人员就是君王；而在有些组织中，他们只是无名小卒。

《明斯特一家》（*The Munsters*）是一部美国系列情景喜剧，从 1964 年开播，一共播出了两季。其笑点在于居住在一个普通市镇模仿鸟（Mockingbird）巷 1313 号的怪兽一家令人捧腹的日常生活。父亲赫尔曼·明斯特是一位愚蠢版的弗兰肯斯坦式怪物[①]，他的妻子是一位吸血鬼，祖父貌似杂耍戏中的德古拉公爵（Count Dracula）[②]，儿子艾迪则是一位小狼人。

荒谬的是，与明斯特一家合住的外甥女玛莉莲却是一位美丽的金发女大学生。但家庭里的其他成员并不认为玛莉莲漂亮，他们觉得她毫无吸引力。他们试图维护她的自尊，但他们总是因为她而感到别扭。《明斯特一家》反映出的一个中心思想是：玛莉莲之所以地位低下，是因为意外地出生在这个古怪的家庭之中。很明

① 弗兰肯斯坦怪物最早出现在玛丽·雪莱（Mary Shelley）的小说 *Frankenstein* 中，是一种人形怪物。——编者注

② 即指吸血鬼。——编者注

显，如果玛莉莲生活在其他正常家庭，她会更加受到欢迎。

很多的开发人员都过着玛莉莲·明斯特式的生活。他们为技术依赖型的公司工作，但他们的工作在很大程度上得不到赞赏，他们在组织中的地位也非常低微。在很多这样的组织里面，经理们掌握着生杀大权。经理负责计划任务、编排进度、追踪进度、估算工作以及给技术人员分配任务。而计划任务、编排进度、追踪进度和估算工作的大权也由各种经理独揽。在他们看来，开发人员就是技术小人物，“不能体会”管理公司工作的真正艰辛。

经理们享受高薪，因为他们负责选择项目，并且把选出的项目作为公司做出的投资决定来运行。技工们只需要埋头干活，按照经理的吩咐一五一十去构建软件即可。如果他们不喜欢这样，他们随时可以离开，经理同 HR 一道，会再找个替代人员。替代人员也许是外包人员，而且也许位于其他大陆，在那里雇用技术小人物的成本远远低于在公司总部的雇用成本。

本模式的另一种变体是销售人员掌握所有的大权。毕竟，（其理由如下）无论产品多好，如果不能卖给客户，它们也是等于零。这两类公司依据的道理就是：开发人员随处可见，并且水平也相差不多，因此他们提供的服务就理应得到尽可能少的薪水。

当然，有些公司对开发人员的态度完全不一样。这些公司认为他们的产品和服务在质量和创新性方面与众不同。他们承认位于前 10%的那些开发人员与普通开发人员在才能和生产率上面有着天壤之别。他们尽全力希望得到最好的开发人员。这产生了一种“开发人员为王”的文化，开发人员在工作量和完成任务的工作方式上拥有绝对的自由度。开发人员常常去整理特性和构建特性，而且通常也是由他们来估算开发的工作量。资深的开发人员跟团队领导的薪酬一样多，有时甚至会更多。通常（但不全是），在把软件作为产品或者产品关键组件的组织之中，开发人员就是君王。

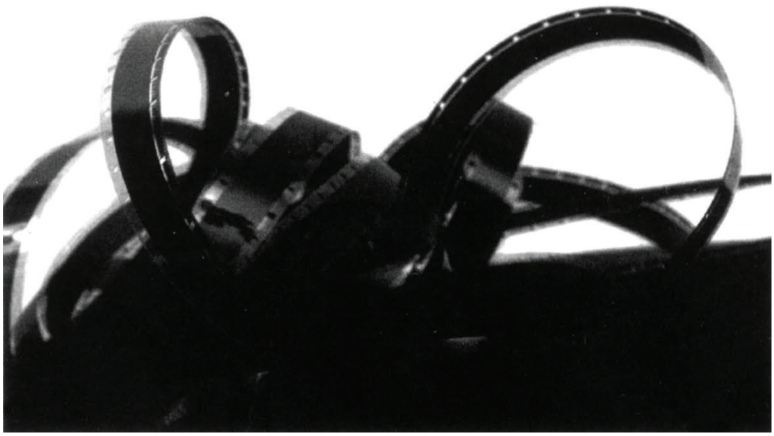
有一种“开发人员为王”的极端文化也会给项目带来障碍。当开发人员不顾及个人决定对其他人的影响，就去优化他们自己的工作或者编排进度表，项目就可能会陷入麻烦。比方说，我们发现有一个项目，其中两位开发人员决定陷软件于不义——遗留大量未完工的类，因为他们知道可以在后期再进行编写——只工作于“有趣的、困难的部分”。在截止期限已然隐约可见的时候，项目团队中的其他人（QA成员和技术文档作者）却只能面对着大量探索性的工作，因为所有的事

情都仍未完成。没有可测的东西，没有可以记录的东西，直到突然有一天所有的事情都完成了——开发人员填好了他们挖下的坑。开发人员优化了他们自己的工作，却使得项目恶化了。

如果你雇用开发人员，有必要先扪心自问质量和创新对于组织成功有多重要。从最高层面来讲，质量和创新都需要真正有才能的开发人员。如果你把开发人员当成一种令你后悔不已，却又不得不支付的成本，并想着尽量降低这种成本的话，你是无法吸引、培养和保留顶级天分的开发人员的。

如果你是一名优秀的开发人员，觉得自己的处境有点像玛莉莲 • 明斯特，请尽管放心，一定会有其他的家庭给予你应得的赞赏。逃离这种不正常的环境吧。

插曲　剪辑掉的底片

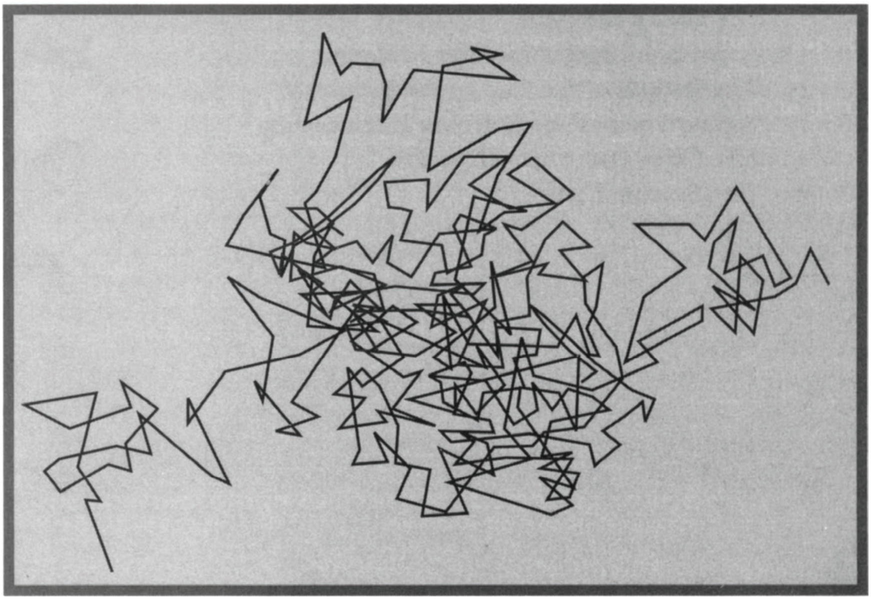


以下这些模式并没有放入到本书之中。

- 如果你的经理是小矮个
- 如果我有把锤子
- ISO 标准避孕套
- 吹着口哨工作
- 公司慈善：不提口臭
- 灵应与功能点：经验主义研究
- 你当法官：他们是用户代表还是潜在的器官捐献人？
- 如果你真的叫呆伯特……
- 没有出口的隔间
- 首次发布时我不会这么做
- 为什么犹大是测试人员的守护神
- 没有白痴还是都是白痴？
- 一条鲻鱼和两条金链
- 毛利战歌：下一个团队建设训练
- 为什么保洁员是唯一完整读到你测试计划的人
- 节约宝贵时间：Beta 版之后再写需求

- 基于信仰的软件工程
- 奶酪永不眠
- 位（Bit）的狂喜
- 午夜抵达印度班加罗尔
- 彩虹尽头的便盆
- 没有需求的软件工程
- HR 聘用我们的天体物理学家
- 蠢人圈

模式 70　布朗运动



在项目愿景尚不明朗的情况下，团队成员就被添加到项目里面。

在新项目的早期，团队的领导承担着压力来完成两件事：定义项目的交付产物和快速取得明显进展。

急于加快发展的心理使得很多经理把项目人数和项目进度等同起来。他们给团队增派人手，尽管他们一点也不清楚新加入的成员能做什么。很自然地，新招聘的人员在很大程度上无法合作顺畅。人太多，疏于指导，最终的结果就是随机活动或者朝任意方向乱动，更像是由罗伯特·布朗（Robert Brown）所观察到的著名现象，即花粉粒在水中的运动。

与本模式相反的形式是项目对于将要做的东西抱有清晰明了的愿景，同时人员配备仍然控制在最少数目以内。愿景规划者把自己与其他人隔离开来，直到他们设定好了项目目标、范围、约束、对交付产品的清晰构想，以及产品会给预料中的受众或者所有者带来什么收益。在项目添加新的团队成员之前，就要完成——明确地完成这些愿景。

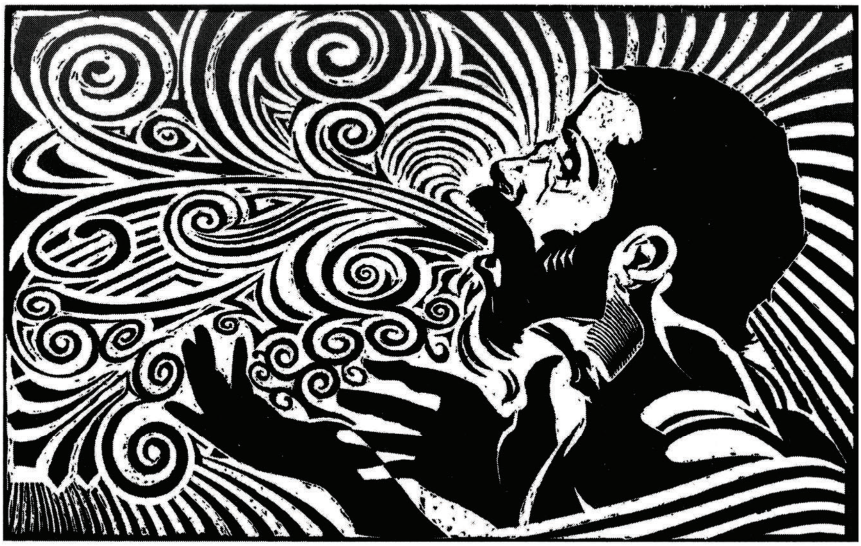
“先把他们送去电影院。至少要等到核心团体精心计划了项目结构，定义了新雇员应该做的事情之后，再把他们加进来。”

——史蒂夫·梅勒（Steve Mellor）

本书作者中的两位曾在一家大型基础设施公司参与了一个长期项目，为对方设计和构建它们最大的应用系统。最终需要的人力是数百人年，但其基本概念和高层架构是由一个仅仅三人组成的团队用三个月的时间构思出来的。类似的例子层出不穷。今天全世界有数千个开发人员工作在 Linux 之上，但 Linux 的构想只是由一个人提出的。构思 C++也只需要一个人，但另一方面，却有整整一个委员会来规划 Ada。

在愿景变得清晰之前就往项目上加人，这只会适得其反。当太多的人试图给项目划定计划，结果就会混乱不堪、逻辑不清。无论最后形成的团队规模多大，清晰的愿景只来源于一个人或者一个很小的团体。

模式 71　大声地、清楚地



大声地、再三地清晰表达项目的目标。

项目的目标就是最高级别的需求和约束。这些目标需要在项目早期就声明出来，并且项目中的每个人都需要不断地去审视。为什么？因为在组织中工作的人们通常会有与项目目标相冲突的个人目标。销售人员希望最大化自己销售的总收入，因为他的提成正是基于利润。产品经理希望最大化自己生产线的利润，这样他的老板才会认可他的成功。工程师希望把所有已答应的功能放到下一个版本，因为奖金就靠着它了。这些目标并不一致，它们之间甚至可能会发生冲突。组织是一个杂乱无序的庞然大物，假如冲突没有显现出来，就没有人会去解决。

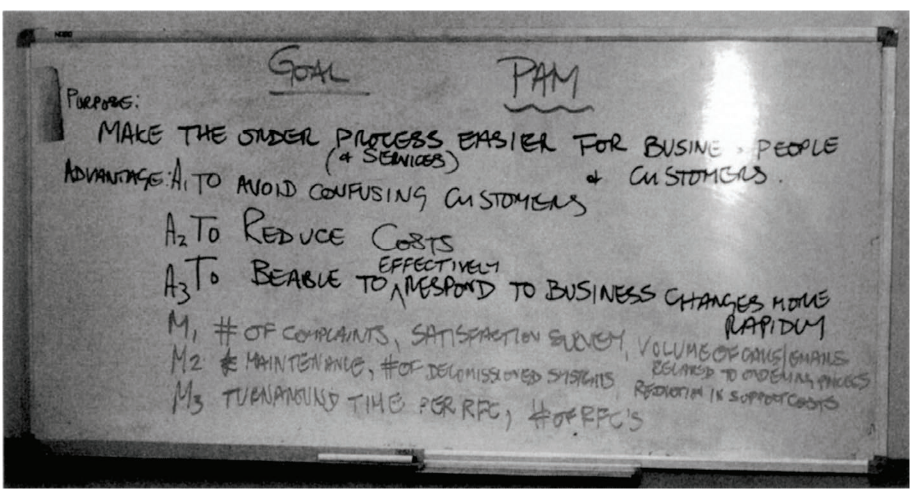
项目不能长时间陷入混乱，否则就群龙无首了。首先，目标需要清晰表达、复审和优化，这样才能保证不同的资助人与利害相干人对于项目的期望真正地汇聚在一起。当项目是要构建一个系统，并且这个系统要跨越多个具有组织级别影响力的独立部门时，在这些不同的团体之间找到一组共同的目标可能是了不起的壮举。

如果利害相干人的确能找到一组无冲突的目标，那这些目标就构成了设计和

建设的愿景。

然而，即使目标定义得当，如果它们不可见，那也是于事无补的。如果不时常拿项目的整体目标来提醒人们，人们很容易就忘记那些目标，忽略项目的主要目的。

> 业务分析师卡罗琳娜与我们的一位客户在开始一个大型项目的时候，把所有的利害相干人召集在一起，帮助他们达成了一组共同的项目目标。然后她把最终得到的目标声明和对应的衡量标准用大写字母写在了一张海报大小的卡片上（见下图）。无论参加哪个项目会议，她都会随身携带，把它放在一把椅子上。只要讨论偏离轨道，她就会拿出项目目标来帮助人们重新聚焦："要么目标是错的，需要重新审视，要么我们就是在偏离轨道。"
>
> 她说两个关键因素使她的方法取得了成功：设定目标的时候每个人都在场，并且在项目的全过程中，每个人始终都能看见这个目标。
>
> ——SQR

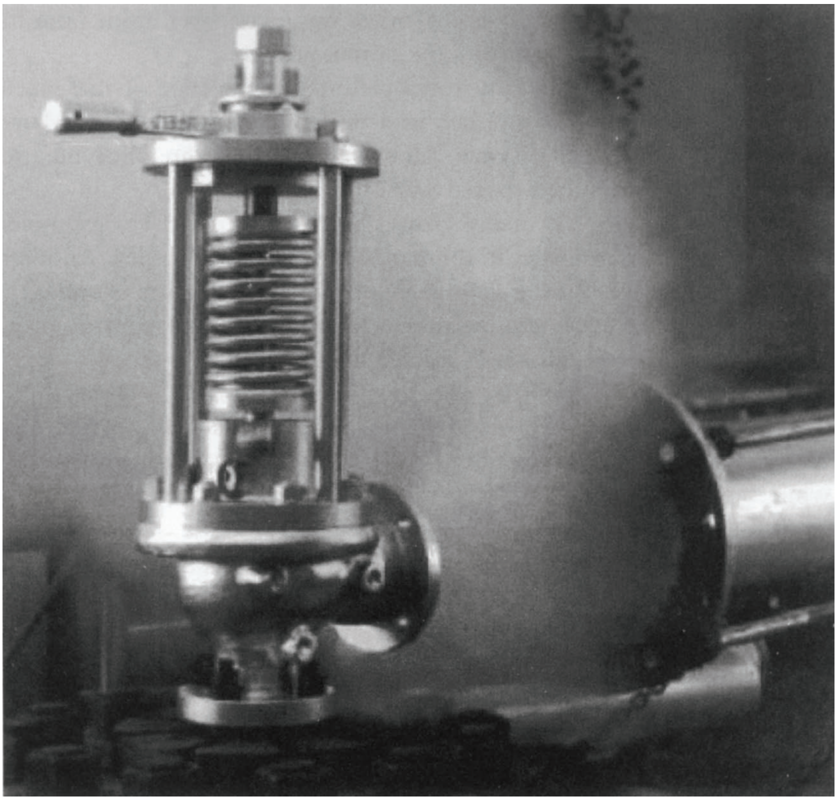


分析人员和业务人员并不是唯一需要持续跟进项目目标的人。设计人员同样需要知道目标，以做出有针对性的设计决策，比如这样问道："你们期望系统的使用年限是多长？如果这个系统总共只会运行三次，那我们的设计与那种至少在接下来十年的每个工作日里都要全天运行的系统完全不一样。"

几乎每一个项目都会面临时间压力，而且随着项目的进行，人们会明显发现并非所有的特性都能在第一个版本里面发布。哪些特性放在第一个版本里面无疑取决于产品和项目管理层的决定，而这些决定正是通过项目的目标来维系的。

拥有正确的目标至关重要。让每个人都始终意识到自己的目标会给项目以及自己开发的产品产生巨大的影响。正如卡罗琳娜那样，你需要在会议桌旁边给你的目标卡片留一个座位。

模式 72　安全阀



为了化解工作中的紧张气氛，团队发明了纾解压力的活动，并演化为团队生活的一部分。

没有人会工作一整天——我们做不到。我们喜欢自己的工作，但是我们都需要周期性地停下手头工作，做些与工作无关的事情来恢复精力。当我们起身下楼喝杯咖啡，我们绝不是在消极怠工——就像拳击手在两个回合之间要暂停一会儿，我们也需要休息一下。

但是我们还需要另外一种休息，它与疲惫或者提神无关。我们需要的是纾解工作的压力。项目上的大部分人都非常努力地、有时甚至是长时间地工作，常常陷入因紧迫的截止期限而带来的压力之中。这很正常，我们也不希望一点压力也

没有。然而，团队不时需要从那种压力中解脱出来。

本模式开头图上的机械安全阀在锅炉内气压太高的时候，会释放蒸汽。很多项目团队创建了他们自己的安全阀——通常是某种与众不同的活动——让他们以特别的方式释放蒸汽。

安全阀有多种形式。有些非常简单，例如，我们知道有一个团队自发地组织了“喷射彩带”的游戏[①]，另一个团队则组织了在办公室隔间之间驾驶微型三轮车的竞赛。

其他的就要复杂一些。某个大型团队定期玩一个名为“秘密刺客”的游戏。每个团队成员都会设定一个“刺杀”目标，同时自己也成为另一个成员的目标。一旦这些秘密任务安排妥当，团队成员就会端着玩具枪，悄悄地逼近自己的目标。他们就像往常一样处理自己的工作，同时又希望自己在被刺客消灭之前先灭掉自己的目标。

这个游戏没有使人们从工作之中脱离出来，恰恰相反的是，公司的一位董事发现玩“秘密刺客”游戏的时候，生产率提高了。游戏的规则也许能佐证这种现象，例如，当雇员坐在自己的办公桌旁时，不能实施刺杀。

另一家公司在每月的第一个星期五，整个软件开发团队就会离开办公室，去观看最新上映的电影（通过邮件投票选出）。令人惊讶的不是人们去观看电影，而是无论看什么电影，也（几乎）无论项目的需求如何，团队中的所有人都参与了。

赌博游戏同样能扮演安全阀的角色。有一个团队赌 UPS 货车的收件时间，猜出最接近的时间的人独自赢得赌金。赌注往往连买一顿午餐都不够，但那种悬念感却总能成功地打破团队工作中的紧张感。

诸如此类的安全阀都诞生于团队内部。有人开始了一项活动，然后这项活动受到了团队其他成员的欢迎。这种活动无法强制推行，也不便由团队以外的人建言。我们看到每当管理层试图为团队打造一个安全阀时，都会遭遇明显的失败。

有一个公司，HR 部门为开发人员搭建了娱乐区——但是没有人使用。一家保险公司空出了一间房间，摆上了沙发和坐垫，贴出了一张告示——鼓励员工在需要休息的时候去使用这个房间。同样地，也没有人使用。开发人员宁愿在巨大的鱼缸前驻足，花几分钟观看鲤鱼嬉戏。

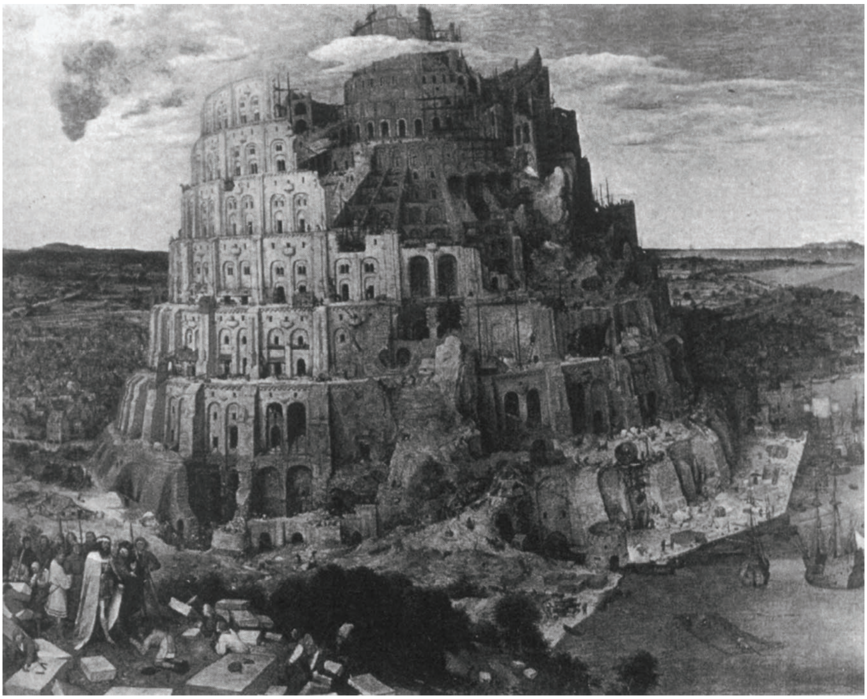
---

① 喷射彩带（Silly String Battle），参与游戏的人向对方喷射彩带。——译者注

这不是说管理层或者 HR 在推动团队选择纾压活动上就无能为力。撞球台、乒乓球台、飞镖靶以及类似的娱乐活动已经在不同的地方被证实非常受欢迎。然而，最有效且最受欢迎的安全阀出自团队内部。

如果你是一名经理，当你看到团队在安全阀活动上面花了一点时间，请不要反对这种做法。也无须鼓励去这样做，因为这是团队自己的娱乐时间，他们最清楚怎么利用这段时间。

模式 73　巴别塔



项目未能开发出一种开发团队和利害相干人都能理解的通用语言。

我们以为别人理解自己——通常来说，这样的假设没有问题。当我们的交谈对象与我们拥有共同的情景和文化背景时，相互之间的交流通常不会出现歧义。在英国的酒吧，当你点一品脱，服务员就会给你递上一品脱啤酒。拿同样的问题去问伦敦的乳品电车司机，他会递给你一瓶牛奶。类似地，当你与坐在一起的团队成员谈网络状态、客户折扣率或者项目使用的其他术语时，你期望你们对其含义拥有一个共同的理解。

正是“别人理解自己”的乐观期望阻碍了你去问自己：“他和我说的是同一件事情吗？我们虽然都在点头，但会不会说的是风牛马不相及？”

巴别塔的故事来自于虚幻想象，它认为组织应该拥有一种通用的语言，而且只要每个项目都使用这种语言，一切就都顺风顺水。但组织是大型的、不断变化

的矛盾有机体，而且诸如服务、订单、资产、策略、客户、雇员和折扣这样的术语在不同项目的上下文中有着不同的含义。与其依靠这些空洞乏味的官话套话，还不如每个项目定义自己的通用语言。通用语言需要有多规范或多严密，取决于其被误解的可能性和严重程度，重则使人丢掉性命，轻则造成少量经济损失，或者使人产生不快。

被误解的风险随着任意一项与个体差异相关的因素而增加，比如领域知识、生活阅历、语言背景或者个性特征。如果你考虑其他的影响，比如地理隔离、安排在多个并行项目之上、外包给其他组织等，这个列表还会更长。很多组织通过收购其他公司变得更大，而每个被收购的公司都有自己特殊的语言，可能需要由新的母公司进行解读和标准化。

项目所需要的语言是一种不断演化的语言，真实地反映了所有的团队成员对问题域的理解。发展这种语言意味着逐步给各种术语提供书面的定义，以反映成员不断深入的理解，同时也意味着让团队中的每个人都非常易于接触和扩充那些术语。

当一个团队开发通用语言，外界对于这些付出可能一无所知。但是在项目内部，参与其中的人愿意认真地、反复地——几乎着迷地——专注于定义术语。成功的团队不会想当然地认为组织里已经存在着通用语言。为了构建项目的通用语言——保护团队不沦为巴别塔，团队愿意提炼、提炼再提炼。

模式 74　惊喜

提供奖赏和奖励的经理听到了意料之外的回应。

在连续六周每周工作六七天之后，版本终于上线了。团队已经疲惫不堪，但每个人都因为工作完成、系统就绪而兴高采烈。在短暂休整以后，团队回到办公司，聚集在项目经理周围，听项目经理宣布胜利的消息以及代表公司致以衷心感谢。为了表达谢意，公司给团队中的每位成员提供了一张来自市区一家顶级餐厅的双人晚宴代金券。随着装有代金券的信封传到每个成员手中，成员之间互相开着玩笑，诸如关于订上若干瓶凯歌皇牌（Veuve Clicquot）香槟酒、烤里脊牛排和整个甜点托盘。等到团队散开，大部分的人重新回去工作，有一个人走近项目经理，说："把这个代金券拿回去吧。如果我妻子认为我在过去的一个半月里面拼命工作，把她和两个孩子扔下不管，就是为了这一顿微不足道的晚餐，她会杀了我。"项目经理无言以对。



代金券这个想法不错。除了那个人以外，对于团队中的其他人来说，那仅仅是一种象征性的东西。它的意思是："去放松一下，给自己挥霍一下。"但是这位成员不那么认为。对于他的妻子来说，那就像是试图进行廉价的收买——大量的工作影响了宝贵的家庭时间，却没有些许的报酬。

组织向团队或者个人送出的奖励几乎不可能做到人人满意，哪怕它们真的只是一种象征，只是为了表达组织的感激之情而已。

当组织习惯于使用奖金和奖赏来诱使行为发生改变，或者去维持一个无法持续的行为（比如每周工作六天），它们只会成功地激发大部分领受者的反抗情绪，

削减他们的士气。给出奖励的组织认为这些奖励可以有效地强化和认可卓越的绩效，但是它们在两个方面陷入了不正常的奖励模式。首先，他们奖励很少的人，这立刻就会在所有没有领受奖励的人心里生出一种不可言喻的情绪，觉得自己被疏远了，受到了不公平对待："我呢？我也是全身心投入在工作上面。我的奖金在哪里？"

其次，奖励这一类东西按理来说是一种令人愉悦的惊喜，却反而变成了一种应得的东西。已领受过奖励的人们期待着、好奇地互相谈论着："你觉得我们在完成项目之后可以拿到什么奖励？"这种奖励没有一点积极的效果。

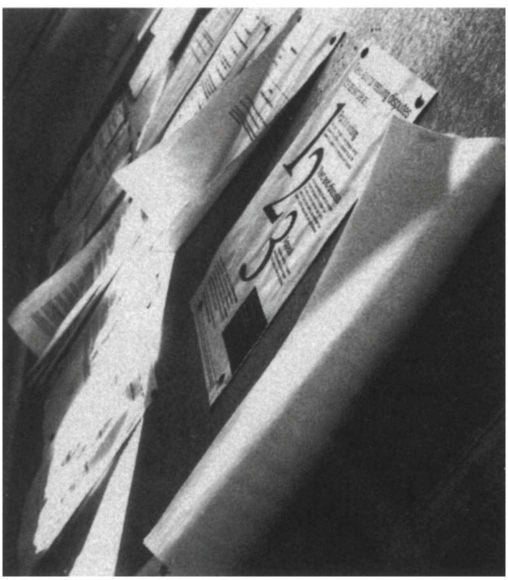
死死抓住奖励和奖金模式的组织从来得不到奖励。

模式 75　冰箱门

团队成员定期地把各自的工作成果展现给团队所有的人。

你经常从经理那里听到“应该知道”原则：只告诉其他人工作中应该知道的东西；除非极为必要，否则不要就手头上的任务交流任何信息；把必须传递的信息流严格限制到最低。

冰箱门模式冲破了这个“忠告”，但却与成功的项目关联紧密。下面是其工作机制。



虽然没有必要让每个人都知道所有的事情，但团队仍然以高度可视化的形式把重要的信息展现出来。这些东西摆放在项目团队所有成员都能看见的地方——在他们进入或者离开办公室/项目作战室的时候，在他们去煮咖啡或者去洗手间的路上。有些信息几乎人人可以更新，他们可以随意地使用备好的铅笔和彩笔做出更新。这些张贴的东西就是不断演化的文档，包含了项目状态和结构的重要信息。

健康的团队在不同的项目角色之间共享以下的产物。

- 版本计划。
- 本周或者当前版本的工作安排，这样每个人都能看到在一个特定的时间段内，所有人都在做什么任务。
- 燃尽图[①]，或者其他（易于领会的）形式的进度报告。

健康的团队通常也会共享如下这些需求产出。

① burn-down chart，燃尽图，也称剩余时间表，显示当前未完成的任务数。——编者注

- 系统的上下文关系图，这样每个人可以对“什么属于系统内”、“什么在系统外”、“接口在哪”一清二楚。
- 用例列表（或者是用例图），在大多数情况下只是一页纸概述了系统将要支撑的流程。这种图通常用不同颜色标识出开发流程的不同阶段，比如用红色代表“已计划”，黄色代表“进行中”，绿色代表“已完成”。
- 包含了用例和领域类型的交叉结构图（一种高层面的、把领域实体和用例对应起来的交叉引用表）。

软件架构通常会展示高层面的软件系统结构——比如，20 个最重要的组件以及相互之间的关系。不要把这个与为市场目标而创作的华而不实的讲演幻灯片混为一谈。这是架构师工作的真正结构图，上面你能看到针对新相关性的手动修改，以及对于不必要相关性的红色问号。

测试人员常常自豪地展示他们测试用例的纵览图和测试用例达到的覆盖率。这样可以让那些不干 QA 工作的人也能很好地理解 QA 和测试如何影响到自己的工作。

诸如此类的可视化陈列品可以供所有的项目成员进行编辑，来展示他们关注的东西，以及他们希望其他人关注的东西。

> “借助于信息辐射器，路过的人不需要询问问题。在他们经过的时候，信息就已经进入了他们脑海。”
>
> ——Alistair Cockburn，《敏捷软件开发》（*Agile Software Development*）（Reading，Mass.: Addison-Wesley, 2002），p.84

项目产物的公开展示反映出团队成员之间的信任。它传达了一种信号——没有什么会仅仅因为主观原因而隐藏起来。没有人会因为让其他人看到了未完成的事物或者进度延迟而担心。冰箱门型项目上的团队成员基本上不会去“偏袒”或者粉饰自己的进度报告。

冰箱门同样帮团队省去了繁重的项目管理和文档分发。很多展示的信息都会在工作之中更新，从而向每个人展现了最新的状态信息。

经过走廊，你也许会看到这样一条提示：“系统跟财务系统的接口实际上是联机流（online stream），而不是离散信息，为了更好地表现出这一点，我们刚刚改动了接口名字。”在墙角，你也许会看到另外一张提示：“你们是否注意到相比于

昨天的冲刺计划（sprint plan），最新的需求变更又要让我们推迟四天才能完成任务？”

冰箱门式的信息展示有一种与生俱来的东西。共享的产物不仅仅展示了达成目标的自豪，而且提供了“团队在这里”的信息，自然而然地就会让所有人的观点达成一致。想想这种做法，再看看在体育激励海报上印上登山运动员、皮划艇队、驰跃的白马以及种种老掉牙却让人热血沸腾的词汇的做法。

你更喜欢在哪种环境下工作?

模式 76　明天会是晴空万里

经理相信未来的平均进度会超过过去的平均进度。

你管理着若干个项目团队。你给自己倒了一杯咖啡，与你的几位团队带头人在一间会议室坐了下来，准备回顾一下当前某些开发工作的进度。议程上的第一个项目对你而言是陌生的——虽然它已经进行了几个月，但最近因为一次重组才改由你接手管理。项目经理比较年轻，但是你已经听说了他的传奇事迹。让我们称其为杰瑞。



杰瑞先带着你熟悉项目的最新进展。他使用了为期三周的迭代，而他的团队刚刚完成三个迭代。在前两个迭代中，团队都没有彻底完成所有计划的工作，而不得不把其中一些工作推迟到下一个迭代。这种趋势有增无减。在这三次迭代之后，团队仅仅完成了前两个迭代的计划任务。(因为优先级的调整和范围的蠕变，某些具体的工作任务发生了变更。)

听到这些，你并不是特别高兴，但也没有觉得完全出乎意料。急切的团队在自己能够完成的工作量上往往过于乐观，尤其是在开发周期的早期阶段。那么，对团队的走向作一些调整也不会太困难。毕竟，距离 Beta 版本的计划发布还有 5 个迭代的开发时间，此外还有 4 个迭代的时间来使系统稳定下来以及响应客户对 Beta 版本的反馈。你只是想知道杰瑞将如何达到这些目标，因此你问道：“按照目前的进度，你打算做出什么调整？”

杰瑞：“实际上，我们非常有信心依照原来的范围和进度安排完成目标。”

你：“但是为了达到那个目标，你需要在接下来的 5 个迭代里面完成原本安排

6个迭代来完成的工作。”

杰瑞：“是的，但我们认为那绝对没问题。我们已经组建了一支非常棒的团队，每个人都对按期交付信心十足。”

你：“太好了，但是你怎么解释前三个迭代里的延误？”

杰瑞：“呃，这个。嗯，那不是由某一个大问题引起的，相反是一大堆小事情——严格意义上说，都是些暂时的挫折。”

你：“说来听听。”

杰瑞：“没问题。嗯，上个月曾经有一次非常严重的断网。这次断网只持续了18个小时，但是我们却花了数天从中恢复。然后达丽莲的父亲突然去世了，她因此离开了一周。你知道，她是我们的产品经理，所以在需求和验收已完成特性的方面，我们都得依赖她。然后在两周前，销售人员突然冒出来，要我们帮助对他们试图达成的庞大交易提供概念论证。那个任务花了3个人数天的时间。”

你：“你说得对。那真是不同寻常。还有其他的事吗？”

杰瑞：“还有其他几件事。你知道夏令时[①]的惨痛事故吗？嗯，那产生了很多有关客户支持的案子，我们的开发人员不得不花时间去处理。而且，哦，对了，杰森因为陪审员义务而被法院征召，并且因为之前他已经推脱了3次，这次法院要求他必须出席。他因此离开了两周，而他又是离线支持特性的主力开发人员。因此，我们不得不把那个特性推迟到更晚。”

你：“哇，真是一件接着一件。基于这些情况，你和你的团队在三个迭代里面做到了当前的程度，还是做得很不错的。”

杰瑞：“谢谢。就像我说的，他们是一支极好的团队。”

你：“那么，在第四个到第八个迭代里面，你留出了多少时间来处理这样一些问题？”

[令人不安的沉默。]

杰瑞：“嗯，啊，我们从没有预料到会遇到像这样一些问题，它们都属于异常情况。你无法预知这样的事情何时会发生。我的意思是，整个夏令时的问题——

① 夏令时，daylight-saving time，是一种为节约能源而人为规定地方时间的制度，在这一制度实行期间所采用的统一时间称为“夏令时”。一般在天亮早的夏季人为将作息时间提前一小时以减少照明量、节约用电。——编者注

纯粹地——是由国会的法案引起的。这种情况要多久才会再发生一次呢？”

你：“那么，你看我理解得对不对啊。你认为你们可以赶上最初计划的日期，而不用砍掉任何的特性，因为你们已经在前三个迭代里面把所有该碰到的倒霉事都碰到了，从今往后，你们的运气就会彻底好转？”

杰瑞不是一名拙劣的经理。他很乐观，乐观到让人感觉有些危险。这或许是出于他乐观的世界观，他不相信众多无法预料的挫折都会降临在他的项目上面。然而，即使杰瑞不是天生的乐观主义者，项目的推动力也会鼓励他表现得像一位乐观主义者。

想想杰瑞更可取的选择。如果他假设项目未来会遭遇平均程度的坏运气，他很可能将不得不同时考虑缩减范围和进度延期。（是的，他可能会寻求额外的资源，但这在短期内对他的项目产生帮助的可能性很小。）这取决于杰瑞如何看待你——他的上级经理——将做出的反应，杰瑞也许不愿意宣称需要做出如此痛苦的改变。

杰瑞之所以不愿意提议缩减特性或者延迟交付，也可能是缘于下面这个事实——在此时此刻，他无法证明他需要这样做。当且仅当一切进展顺利的时候，计划的工作量才能勉强与剩余的时间相一致。如果杰瑞为了给未来的坏运气预留富余量而去要求缩减特性，你或者其他人又对这种做法的必要性提出质疑，那么，由于事情还没有发生，杰瑞就无法拿出任何确切的问题来佐证缩减特性的合理性。

极限编程提供了一种优雅的方式来抵制这种乐观情绪，它称为“昨日天气[①]”。下一个迭代的生产率（计划生产率）被假定为不超过上一个迭代（刚刚结束的迭代）的实际生产率。无论你是使用“昨日天气”还是自行调整，你必须要知道的一件事是项目未来的坏运气不会完全消失。因此，请相应地制订计划。

① Kent Beck 与 Martin Fowler 合著，*Planning Extreme Programming*（Boston：Addison-Wesley，2001），第8章，第33~34页。

模式 77 堆积



利害相干人宣称支持项目，然而却一直百般阻挠直到项目失败。

在美式橄榄球中，“堆积”是一种惩罚，指的是防守队员跳起来压到已经倒下的持球队员身上。壮硕的边锋重重地落在持球队员的背上，是想向他传达一个信号，以防他胆敢再次持球闯入他们的疆域。“堆积”这种做法属于犯规。

项目工作的堆积通常表现为给产品增加无关痛痒的特性，而这些特性的成本/收益比尚未可知。虽然看上去富于建设性，但这种行为的隐秘目标是增加死亡的可能性。这也是彼得·金（Peter Keen）在其论文中称之为“反实现”的变体。在他的这篇经典论文[1]之中，金描写了饶有趣味的观察现象，那些想要挫败新项目的人没有必要冒着风险真正站出来反对。恰恰相反，他们可以通过提议数十个能够“帮助项目达成卓越目标”的附加特性和改进措施，给项目以最积极的信任投票。

采用大量迭代的项目团队对于“堆积”并不具有免疫力，但是他们的确拥有

① Peter G.W. Keen，“信息系统和组织变更”，*Communications of the ACM*，第24卷，第一期（1981年1月），pp. 24-33。

天然的和强大的防御工事：在计划每个迭代的排列顺序时，他们强制按照从关键到“堆积”的级别来评估特性，并相应地指定优先级。早期的实现包括了关键级别的特性，其他级别的特性则被添加到列表的末端。如果添加的下一个特性所承诺增加的收益少于增加的成本，项目或许就会宣告完工。因为所有重要的部分在很早之前就已经交付，项目没有必要再继续了。

“反实现”的种种变体（你的确需要读读金的论文）非常普遍，如果你识别不出的话，那是因为你观察得不够仔细。

模式 78　变更时节

在项目的整个过程中，范围变更的时机只出现在特定的时刻——通常是开发迭代的开始或者结束阶段。

软件开发项目需要我们作出一连串的选择。有些选择必不可少，有些则影响有限。对于前者，我们在开发阶段所做出的一项影响最为深远的选择是定义范围：哪些属于范围之内，哪些属于范围之外。

决定项目范围就是“第 22 条军规”[①]：你需要尽早地得到准确的范围值，但随后你几乎总是要做出调整。最终，由于想要完成工作，你对于范围变更的容忍耐心消磨殆尽，正如图 1 所示。



木版画，Bobby Donovan 作品

如果你领悟了这幅图的含义，你也许能推断出项目范围在项目的整个过程中会不断变更。虽然变更可能发生，但大部分的项目经理都认识到接受所有出现的范围变更是不现实的。为什么？因为范围变更会产生混乱。它们对人们的日常工作可能产生严重的影响。人们根据变更的内容去重新评估手头做的事情——这减缓了项目的速度。

为了在改进范围的需要与保持前行势头的需要之间取得平衡，很多团队把项目的开发过程分为多个短迭代，每个迭代都严格限制范围变更。第一个迭代使用一开始定义的范围，直到这个迭代结束，这个范围不能变更。在此期间，做第二

① Catch-22，美国作家 Joseph Heller 于 1961 年写成的一部讽刺小说。在这里指的是逻辑上相互矛盾的、左右为难的事情。——编者注

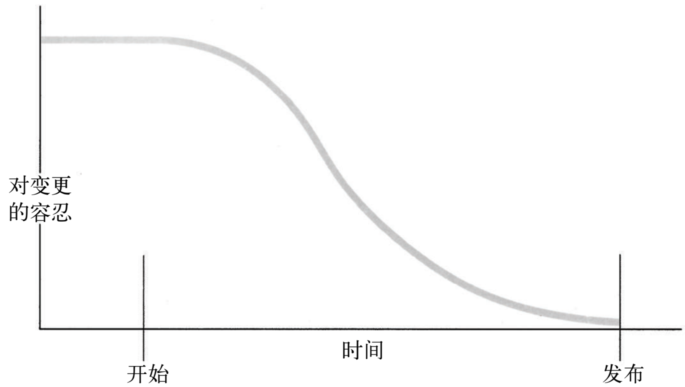


图　1

个迭代计划的人们也许会考虑范围的变更。但是在每个迭代内部，开发人员和团队的其他人员不会被打扰。

本模式看上去如图 2 所示。

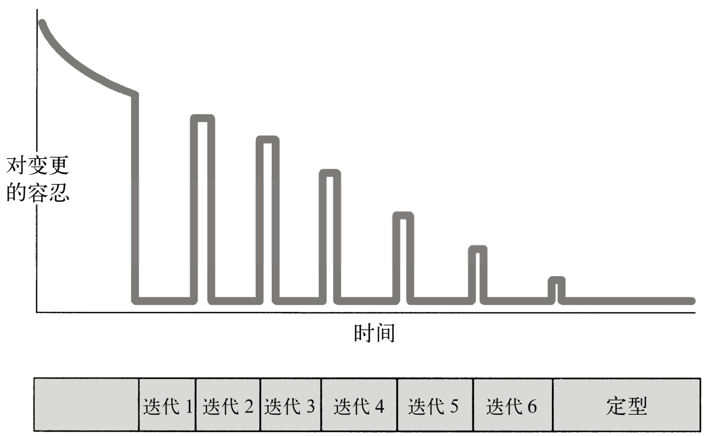


图　2

值得指出的是，只有当迭代持续时间较短时，本方法才能奏效。在更长的迭代（比如说，为期 12 周）里面推迟所有的需求变更，不一定是明智的，而且有时甚至根本不可能推迟。两到六周的迭代既降低了范围变更的破坏作用，又不会限制项目本身的发展。

模式 79 造纸厂



组织通过迄今产出文档的重量和数量来衡量进度。

项目就是一次旅程，在这个旅程中，人们去理解问题，直至足以得出满足既定约束集的解决方案。而且随着理解越来越深入，方方面面都需要与不同的利害相干人进行沟通。针对这些理解的沟通常常既要使用纸质文档，还要使用电子文档。在设计每次沟通的时候，你需要问这些问题：*你试图沟通的内容是什么，什么是最有效的沟通媒介？*这两个难题不解决，意味着人们的沟通最后会以太多、太少或者错误的信息而告终，根本无法提供必要的反馈。

下面这些说法听起来是否耳熟？

- “我们必须在这周结束之前完成可行性报告。”
- “功能规格书必须在下周二之前就绪。”
- “我们必须把这次会议的会议纪要分发给所有的利害相干人。”

如果你对这些话回问“为什么”，答案可能是“因为我们必须在我们项目的这个阶段产出这份文档”。如果你进一步追问：“请精确一些，文档应该包含什么？文档的目的是什么？谁又会使用这份文档来做出哪些决策？”然后，你就会发现人们变得瞠目结舌。他们之所以产出文档，仅仅因为那是接下来要做的事情。

如果你在项目里面发现了这种行为，那么你们也许正工作于“造纸厂”之中。

在“造纸厂”之中，每项活动都是以文档的产出作为标识的，项目的进度也是以产出文档的数量——而不是文档所包含的内容——来衡量的。“造纸厂”的原则是：为防万一有人需要什么，让我们把所有的一切都给所有人。

“造纸厂”的另一种形式是，不同文档的内容相互之间没有形式上的关联。例如，同一个流程在两份文档中的名字可能非常不同或者稍微不一致。这样，虽然人们猜测这两者是同一样东西，但文档并不具备形式上的可溯性，反倒充斥着大量的假设和混乱不堪。本模式的另一个迹象是人们变得痴迷于占有文档。如果你产出了某个文档——不论内容是什么——人们就会问：“我能拥有一份副本吗？”每个人都希望拥有每一份文档的副本。

“造纸厂”是有害的，因为一旦人们去关注产出文档的重量，他们就会停止思考更为重要的事情：我们是否在从事有助于实现项目目标的工作？

非“造纸厂”的项目使用团队内部达成一致的客观衡量标准，比如输入和输出的数目、业务流程、用例、约束、特性、工作代码的模块、功能点、数据元素，或者其他适合于项目的东西[①]。

这些项目团队没有去自动生成文档，而是考虑使用其他的方式来沟通进度。他们使用白板、电话会议、博客和原型作为沟通媒介。而且，他们也把所有的项目制品保存到中央项目库，并且让所有需要这些制品的人都可以自由地得到他们，从而阻止了人们去收藏一个个文档。

关键在于每一份产出文档都应该满足某些明确界定的要求，而且其包含的内容应该在整个项目的知识库中都可溯。

① 欲了解关于“按照项目真实的需求裁剪流程和文档”的更多信息，请参阅第12项模式。

模式 80　离岸荒唐事



领导们被低廉的工人薪资所吸引，启动了离岸开发计划，使得在各个开发地点之间沟通的难度剧增。

要想在软件经理之间激起争论，一个屡试不爽的方法就是提起离岸开发的话题。在过去的 15 年中，尽管一直饱受着争议，离岸软件开发和离岸技术支持的应用已经从边缘的时尚变成了行业的主流。有些经理把离岸开发视作未来的必然之路，但有些人则视之为愚笨无能的成本削减人士孤注一掷的举措。还有一些人把它看成是可能会有用的工具，认为它虽然带来了巨大的挑战，但这些挑战还是可以克服的。差不多所有人都认同的一点是很多途径都会导致离岸开发产生可怕的后果。

下面是一些我们最认同的管理“顿悟”——它们给后续的离岸开发冒险宣判了死刑。

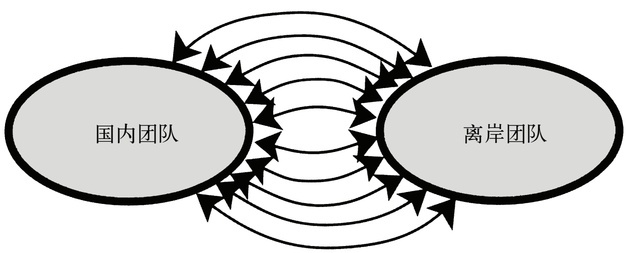
- “从今以后，一旦国内团队有人离职，你将不得不在法兰士约瑟夫地群岛（Franz Josef Land）[1]招聘一个员工来代替他。”

① 位于北冰洋的群岛。——编者注

- “看来我们低估了这个 portlet 组件的工作量。我听说安道尔有三名开发人员在接下来的两个月里面不会太忙。我们让他们来帮帮忙吧。”
- “噢，下一个版本的进度安排看上去非常紧张。凑齐足够数目开发人员的唯一途径就是离岸开发，那会帮助我们快一点完成。”

这些论调显示了人们没有认识到不同地点之间沟通与迭代的高成本。近乎随机地（例如，为了接替离职的人）把工作分派到不同的地点，会极大地增加不同地点之间需要的最小带宽。同样，如果给第一线员工安排任务和提供反馈的经理位于其他的地点，两地之间所需要的沟通频率和强度也得大为增加。

这里的基本原理并无新意。为了使系统的复杂度变得可控，人们长期教诲我们去分割系统，从而尽可能降低主要子组件之间的接口复杂度。相同的概念同样适用于团队工作流程的设计。如果团队包括了位于多个地点的人群，特别是如果有些人位于不同的时区，增加的成本就非常可观了。我们希望减少对高频率的高带宽的沟通以及跨洋迭代的需要。我们希望避免下图的情形：



如何判断自己是否已经纠缠于离岸荒唐事？看看有没有下面这些“症状”。

- 整个开发周期里面，在每天早上 6 点（或者 8 点）召开例会，这样位于不同地点的开发人员可以同步各自的进展。
- 3 个位于国内地点的人试图管理 2 个离岸开发人员的工作。
- 第一线员工的直接经理位于 4 个时区以外的地点。
- 离岸团队为六个产品开发特性，但自己却无法发布任何特性。

虽然指出了以上种种，但我们并不想劝阻你使用离岸的开发资源。自上世纪 90 年代以来，我们已经有过离岸开发的成功经历。为了充分发挥离岸开发的潜力，不妨听听有经验的人给出的以下建议。

(1) 本地迭代。只要可能，就把需要快速迭代的工作的各阶段任务分派给单一地点的团队，无论是国内还是离岸。

(2) 要认识到最初几次利用离岸开发会比近地开发花费更长的时间。团队需要时间来锻炼和适应，而离岸开发的成功需要这些锻炼。

(3) 意识到离岸团队与国内团队在大多数方面没有任何区别。他们渴望接受挑战性的、有意义的工作。大部分离岸点的人力市场已经变得非常富于竞争力，这些市场上的顶级软件工程师拥有很多的选择。他们喜欢令人兴奋的、可以让他们提升技能的工作，并且会逃离那些仅仅从事维护工作的血汗工厂。

(4) 树立各个地点的目标。每个地点都需要精神层面的东西。虽然有大量的因素都可以区分出健康的地点与不健康的地点，但活跃地点的一个共同特征就是它们都有着明确的使命。它们构建一个特别的产品或者系统，抑或构建大型产品或系统中的某一项重要的、数得上名的部分。相反地，如果某个地点包括了大量无足轻重的开发和支持团队，却没有明确的共同目标，它们往往会表现得低落士气。在创建一个离岸开发地点时，首先想想你的离岸团队将如何认同该地点的使命。

这些步骤——特别是最后一个——同样也能帮助你避免本地的荒唐事。

模式 81　作战室



使用专用的作战室，把项目列为重点。

年复一年，我们很少碰到有项目拥有自己专用的作战室——用工作产物装饰墙壁、项目成员在项目的公共空间里面互相协作。虽然这并非主流，但该项模式仍然值得分析，因为这些项目的成员为了成功往往会拼命努力。

> 我慢慢认为不值得拥有作战室的项目也许根本不值得去做。
>
> ——TDM

作战室表明了一种态度——大量面对面的交流对于项目的成功至关重要。此外，它强调了积极地展示工作的产物对团队的凝聚与工作的引导尤为重要。最后，它也清楚地展示了利害相干人为了项目成功而大力投资的决心。“房产”是一种典型的投资，而且配备真实“房产”的项目传达了一个强有力的信号——项目很重要。

在大多数情形下，作战室只是一个被征用的会议室。房间足够宽敞，可以容纳项目的全部成员，还能给少量的参观者留下一些空间。在大多数的日子里，在每天的某个时间段内，项目成员常常会聚集在作战室里面或者周围，他们也会在作战室里面进行大部分的关键接口讨论，并召开设计与再设计会议。

这些持续展示的产出物包括未完成的交付物、进行中的设计、进度安排、PERT 图和人力负荷图、风险列表、工作分解结构、工作产物的组合以及其他管理上的产出物。每当团队成员想要找出一项重要计划或者设计产物，以及查看队友们的贡献时，他们就会转到作战室。（第 75 项模式对此有更为详细的描述。）在最好的情况下，每一位工作者在毗邻作战室的地方都会有个人的工作空间，这样，作战室和周围的环境就共同形成了一个明确界定的项目区域。

项目经理频繁地占用作战室，在那里运筹着项目的运转。因为展列的某些产出和工作产物是由项目经理生产的，所以分析和更新它们便是作战室的任务（这些任务会自然而然地聚积到经理那里）。

有一点是显而易见的，但再强调一次也不为过：仅仅声称项目拥有作战室，而且为之留出空间并不能奏效。关键在于如何把作战室变成项目不可或缺的有机组成部分。作战室必须是项目自身选择的一种管理举措，只有这样，作战室才能产生*魔法效应*，因为它并不是在任何情况下都那么神奇。

模式 82　什么味道

组织中的人们无法察觉隐藏于表面之下的究竟是活力还是衰败。



- 福勒太太的面包店位于法国阿尔卑斯的夏木尼（Chamonix）小镇。每一个清晨，每一位进来买新鲜面包或者羊角面包的顾客都会情不自禁地微笑：这个地方闻起来如此香甜。虽然福勒太太和她的员工向愉悦的顾客提供了优异的服务，但是他们的脸上却见不到笑容。
- 长岛的东头有几家鸭场，这些鸭场大部分都是私家拥有以及私家运营。无需看到鸭场，也无需看到任何标识，光闻气味你就能知道自己正在走近鸭场。恶臭就是对这种气味的绝佳形容。如果你可以忍住气味，靠得更近一些，你的眼睛会不自觉地流泪。那些家庭如何能够日复一日地在鸭场里劳作？唯一可能的答案就是他们闻不到那些鸭子的排泄物。

作为一种隐喻，项目和整个组织的气味通常都很强烈，这些气味各不相同，有面包店的甜香，也有鸭场的恶臭等，但是项目成员的四周都充溢着这些气味。

所有的员工都需要知道他们的组织闻起来气味如何，从而可以决定做出以下哪种反应。

- 深深地呼吸，保持原状。
- 打开一点窗户。
- 用烟熏消毒。

无论你在组织内处于何种位置，你都无法依靠自己判定组织的气味。你需要从外部世界引入一些新的鼻子，在你的周围四下里闻闻。有趣的是，如果你在多个项目上待过一段时间，你或许就完全有能力为其他的组织做同样的工作（闻项目的气味）。你或许能够借助某个专业协会的本地分部，启动一个名为“现在就闻：一切为了事实真相[①]（SNIFF）”的计划。

我们已经在不同的组织里面闻了很多年。不多加解释，以下是我们曾经闻到的不同于面包店和鸭场的真实气味：

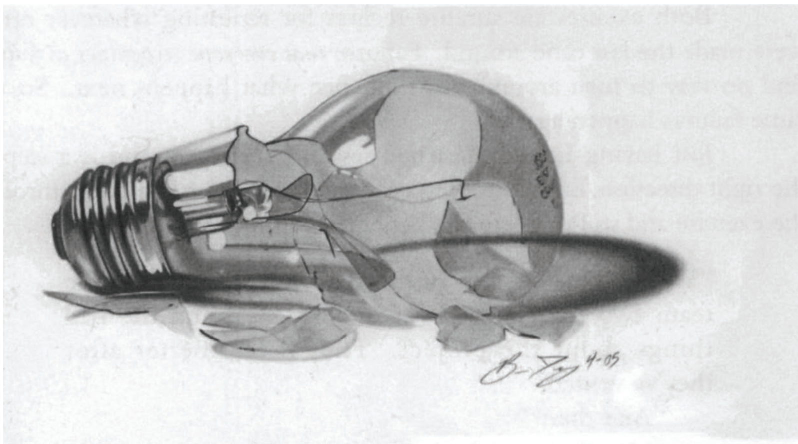
- 哺乳室；
- 猴子笼；
- 青少年的卧室；
- 后台化妆室；
- 海风；
- 肿瘤专家的等待室；
- 发霉的味道；
- 运动场；
- 电火花；
- 雪茄吧。

如果你不能根据组织的气味猜测出组织的类型，那么，或者你已经身处其中，或者你从未踏入过这种组织的办公室。

正如 Lynyrd Skynyrd 乐队在歌中所唱：“你难道闻不到那种气味吗？”

① 本处的英文原文为“Smell-Now: It’s-For-Facts”，简写正好是 SNIFF，即用鼻子闻。作者在这里巧妙地双关。——译者注

模式 83　不从教训中学习



Brian Duey 所做的铅笔画

团队认识到自己的错误，却又一次接一次地重蹈覆辙。

哪些活动在你的项目里是重要的？

> 编写代码？绝对重要！
>
> 整理出正确的需求，基于需求测试产品？呃，是的！
>
> 设计？呃，算是吧。
>
> 在项目结束之后召开"经验学习"会议以改进工作方法？召开什么？经验学习？为什么我们要把时间浪费在这件事情上面？对于上一个项目，我们已经无法改变什么了——它已经结束了。（终于结束了！）而且，我们没有时间来幽思冥想，下一个项目的款项已经到位，我们最好立即开始行动。

如果以上的反应在你的公司里面非常普遍，也不是只有你一家公司是这样。

在项目之后不去检视项目成败的一个借口是自认为时间不充分。另一个借口带着嘲讽的语气："钻研我们过去的错误有什么好处？公司不会允许我们改变任何东西。而且，我们已经接近于 CMMI 3 级了，我们需要坚持现在的做法，否则我们可能会倒退回 2 级。"

要想重蹈覆辙，这两个借口可都是屡试不爽的万金油。对于每个人都承认的失败，如果不想办法去改进、去汲取教训，那么，同样的失败会一再发生。

在项目结束的时候召开经验学习会议是朝着正确方向迈出的一步，但它还不够。很可能即便仔细检视了以往的行为，却依然无法获得回报。想想如下的交流。

> “我们这里会对大部分的项目召开这样的会议。给团队 2 个小时来回想一下项目之中做得好的和不好的地方。发泄之后，他们感觉会好一些。”
>
> “然后呢？”
>
> “嗯，然后每个人接着进行下一个项目。”
>
> “继续做与他们之前做过的完全一样的事情？”
>
> “呃，差不多吧。”

这怎么可能？回顾怎么可能最终变成了释压机制而不是变革的催化剂？毕竟，这些回顾会议明明是作为“变更的机制”才合理存在。这听上去没有问题，但这个合理性缺少了一个词：它们事实上的本意是“内部变更的机制”。难就难在这里。

项目结束之后举行回顾会议的项目往往发现自己强调的问题，在严格意义上都不是项目内部的问题。这些问题根源于外部给项目强加的约束：组织接口、超越人事分工原则的不充分接触、人为设立的中间人、强行的人员配备不足、早期配备的人员过多、阻碍团队的强制标准、疯狂的进度安排、缺少文员支持等问题。因为这些问题都超出了团队的范围，它们的解决方案可能会被认为是僭越了职权。

虽然我们忙得焦头烂额，但要把完全处于团队范畴之内的变更落到实处，其难度很可能非常大。因此，这是一个进退维谷的情形：外部的变化很难，而且因为人事权力上的原因让人提心吊胆；内部的变化很难，而且因为操作上的原因让人胆战心惊。怪不得回顾有时变成了单纯抱怨的会议，也没有严肃的意愿来改变任何事情。

当经验学习的过程主要是以发泄而结束时，你会发现公布的结果都是人们的观察结果，而不是行动事项。虽然在操作上扭转这种局面很容易，但要成功地扭转局面却并非易事。关键在于坚持为每一个明确的问题（不论是团队权力范围之内还是之外）制订具体行动方案：把找出来的每一件没有帮助到团队或者曾经阻

碍过团队的事情，与将在下一次迭代或者版本（又或者下一个项目）中付诸实践的某一项行动事项关联在一起。这样就把经验学习的团队带入了建设性设计的模式。虽然有些行动方案涉及组织的问题，但是也不能太离谱。它们与团队设计的内部项目流程一样，都必须通过可执行性测试。

开这种会时，还有一个操作上的小技巧，那就是让每个人最少准备关于项目的一件好事情和一件坏事情出席会议。（这容许了参与者抛出可能被认为僭越职权的话题。）一个相似的技巧能够帮助团队在回顾时成功地激发变更：坚持至少有一项行动方案完全属于团队范畴之内，并且至少有一项行动方案是部分或者全部属于团队以外。

每一项行动事项都应该指出将如何修改某些方法、任务、人机界面或者强加的约束以避免下一次出现同样的问题。行动方案必须明确，而且它们必须拥有一个特定的目标指引变化的进行，让变化落实到那个目标。因为行动方案将需要做的改变限制在指引领域，因而更有可能被外部的权力所接受。

经验学习会议常常是在项目的结束阶段召开①，但是在每次迭代或者每次发布的结束阶段，主持中期的小型回顾同样也很有必要。

从经验分享中获益的公司都是无畏的：他们让自己的方法、流程和组织结构去接受严格的检查——哪怕这样做会受到尖锐的批评——而且，他们真正地愿意做出改变。为这种类型的组织工作是一大乐事。参与者发现，有机会去推动变化对于公司的文化特别宝贵。

---

① 经验学习会议也被称作项目回顾、尸检或者后项目回顾。对于“如何主持你的经验分享会议”，请参阅 Norman L. Kerth 的 *Project Retrospectives: A Handbook for Team Reviews*（New York：Dorset House 出版社，2001年）获得宝贵的建议。

模式 84　不成熟的想法神圣不可侵犯



团队愿意鼓励、呵护即使看起来不成熟的想法。

一旦团队成员不情愿讲出乍看起来显得不成熟的想法，进度就可能会慢下来，有时甚至会完全停止。优秀的团队让人们可以安全地讲出那些不成熟的想法，很多团队也鼓励这种行为。如果原始的想法不完美，它可以被改进，但团队允许人们将这些想法陈述出来。

不成熟的想法是项目生命周期的一部分，团队应该视之为需要保护和培育的东西。举例来说，只有当团队成员对脱口讲出他们头脑中的想法——无论这个想法看起来如何不完备、直觉上如何不可能或者欠考虑——感到安全时，头脑风暴会议和其他创新性的研讨会才能发挥效应。此外，他们可以大胆地讲出他们脑海中的东西，根本不用担心会受到指责或者嘲笑。经验告诉我们，即使是最不成熟的想法，只要受到团队的尊重并允许其存在，有时也会转变成有价值的商业产品。

允许陈述不成熟的想法可能需要行为上的改变。有些人和团队，同样还包括组织，习惯把所有不是那么直接或者明显可行的想法都统统丢掉。任何想提出一个想法的人都不得不三思，先确保想法是无懈可击的，然后再以每个人都能清楚

地明白该想法价值的方式讲出来。如果有哪一点没做到，这样的建议必然会碰一鼻子灰。强迫所有的想法必须是完全成型之后才能公开述说，这样的组织等于是否定了团队提升的益处，遏止了原本应该稳定持续的项目创新流。大部分的想法都能从集思广益、群策群力的优化之中获益。

相对于发明新东西，人类更擅长改善已有的东西，而且几乎所有的想法都可以被优化——如果你能坚持下去。[①] 可以理解的是，并不是所有的团队成员都是伟大的发明家，而且不是所有的人都像自己所期冀的那样表达清晰。试验性的想法可以加以讨论，有时甚至是讨论得脸红脖子粗，通过团队讨论，想法变得更加成熟、更为可行。当然，不是所有的想法都能通过讨论而被采纳，但是每一个想法都应该获得机会。

想法不应受到约束。除非时间极其短暂，否则为什么要急着摒弃那些不是立即可行的想法？团队所需要做的同样也是不加约束，形成一种使得团队成员觉得能够提议不成熟想法的文化。

① 詹姆斯・戴森（James Dyson）的不成熟想法——离心真空吸尘器，经过了 15 年和 5 000 多个原型才转为商业产品。One Dyson 模型现在是美国、英国、爱尔兰、西班牙、比利时、瑞士、澳大利亚等国家最畅销的真空吸尘器，而另一款模型是在日本畅销的真空吸尘器。

模式 85　渗漏



时间和金钱往往会从衡量密切的范畴“逃离”到衡量不那么密切的范畴。

这真是一个进退两难之境：你正忙于优化服务端的代码，想把公司新门户的一小块界面修改得更为赏心悦目。虽然你在这该死的事情上又投入了 15 个小时，但它仍不能让人满意。正当你纠结于代码时，项目管理办公室（Program Management Office，PMO）的一个家伙走了过来，喋喋不休地询问你这周的时间记录：“请立即登录 PMO 系统，填上数字。”在填写时间记录的时候，你意识到对于任务“5321：实现动态门户接口”，你之前安排的活动已经差不多把时间用完了。如果再加 15 个小时的时间，就会把分配给该项任务的时间都消耗殆尽，而且必然会有人问你：“任务完成了吗？”当然，任务没有完成。周围人人愁眉不展，下一个问题肯定是：“好吧，什么时候能完成？”想想就不寒而栗，你可不愿意沦落到那个地步。

幸运的是，在工作分解结构（WBS）里面还有一项任务尚未登记——“5977：优化响应时间”。现在再回头看看过去 15 个小时里面你一直在做的任务（你不必太严格地去看），难道它看上去不像是在通过实现动态的门户界面来对响应时间做

调试？没错，这两者足够接近了。你决然地把这 15 个小时用于完成任务 5977。大部分的任务分解结构都非常含糊，足以给工作时间汇报留下一定的余地。

管理层紧紧地盯着那些差不多到期的任务，因为当分配的时间耗尽的时候，工作就应该完成。没有人会去注意其他的任务，因为它还远在“明日[①]”之后。这种无恶意地把 15 个小时从一项任务转移到另一项任务就被称为渗漏。

大部分的项目上存在着两种不同类型的渗漏：第一种是在工作完成的时候，把它归到了错误的范畴，正如上面的例子；第二种是把工作的一部分推迟成后期的任务。如果工作描述得非常严格，第二种类型的渗漏是不可能的，但至少对于某些工作，通常都有足够的余地从早期的任务推迟到后期。这两种类型的渗漏都会导致相同的后果：无形之中给项目带来了延期。或者增加了工作，或者缩减了项目的后期活动的可用时间，这些都使得项目更加难以按时完成。

想想以下常见的渗漏例子。

- 那些时间和人工都已经消耗殆尽的任务比那些尚有时间的任务被更严密地监视，因此工作从前者渗漏到后者。（这就是第一类渗漏：错误归类。）
- 那些不久将达到预定日期的任务比那些更晚达到预定日期的任务被更严密地监视，因此往往发生一些渗漏——工作转移到后面的任务之中。（这是第二类渗漏，因为部分工作被推迟了。）
- 需求分析的某些工作渗漏出来，由于还要编码或者测试，需求分析的工作只能草草了事。（第二类渗漏。）
- 由于计划的创新性活动往往定义得很含糊，它们有时会被渗漏下去。（又是第一类渗漏。）
- 往往在困难的工作之前完成容易的工作。这一种工作渗漏是从紧密关注的元类别“看看多少已经完成”渗漏到略微含糊的元类别“还剩下哪些事情需要做”。
- 工作从整个项目之中渗漏出来，进入到项目之后的维护活动。

你也许会认为渗漏纯粹是统计上的问题，只有那些试图编制切合实际的项目工作概况的人才会感兴趣。但是渗漏对于项目的后果可能更为隐性：它可能导致部分（有时甚至是整体）失去控制。当工作从整个项目之中渗漏出去，后果就是

① 请参阅第 7 项模式。

交付之后仍需要修复再修复的劣质产品，管理层也因此失去了对产品质量的控制。而且，当困难的工作从早期活动之中渗漏出来后(所以才能按时完成早期的活动)，难度的密度分布就随着时间不断累加，最后陷入那句常说的至理名言——项目最后 5%的工作花去了远远多于 5%的时间。

“大多数人理解金钱的时间价值，但不理解时间的金钱价值。”

——Steve McMenamin

模式 86　模板僵尸

方框里写字了吗?



这个方框呢?



项目团队使用模板——而不是对于产品交付所必需的、经过深思熟虑的流程——来驱动自己的工作。

当你发现项目团队将精力集中于产出标准的文档，而不是去仔细考虑文档的内容时，你就已然身陷于模板僵尸的世界了。这种对于填满空格活动的痴迷表现为下面的质量检查。

> “我已经完成了项目启动文档。”
>
> “你怎么知道文档完成了？”
>
> “因为我在每个标题下面都已经写了一些东西。”
>
> “哦，太好了。现在我们可以让客户签字同意了。”

在模板僵尸的世界中，格式为王，没有必要对文档的内容进行思考。真的，一点都不用思考。重要的事情是每个规定的标题下面都有一些东西——不论是什么。毫不为奇的是，模板僵尸都精通于剪切、粘贴以及忽略掉所有与模板要求不符的东西。

我们并不是说使用模板一定不好。事实上，它们——特别是检查列表和问题框架——提供了一个非常好的途径来传承经验。但当模板僵化得如同铭刻在石头

上一样、组织设想每一个项目都与此前的项目一模一样时，问题就出现了。模板僵尸们相信只要能把任何东西放入模板的所有区块里，他们就一定能成功。模板僵尸不是去直面棘手的现实——每个项目都是不同的，也不是只把模板视为指导项目的工具，而是屈从于把自己大脑掏空、把空白处填满的诱惑。

> 在我参加的一次回顾会议上，团队成员在讨论一个设计想法。有人以模板没有规定收集那个想法为由表示反对。那个想法应该是在后期的另外一份文档中包括。这个团队没有去变更模板，而是简单地拒绝了那个想法。
>
> 在另一个组织中，团队反对这个理由了。当局者强迫团队应该使用一组由外部方法学专家草拟的标准模板。该模板是一个最糟糕的按部就班的例子。遵循这些条条框框的下场肯定是失败。团队成员开始了秘密行动，使用能让他们完成实际工作的方法。然后，为了满足方法学专家的规矩，他们雇用了一名抄写员来填写那些委实无聊的模板。没有人去读这些抄写出来的文档模板，但是却让领导感到满意，因为它们已经有了足够的页数。
>
> ——SQR

如果你发现自己确实困扰于总是把某些东西填到模板的各个标题之下，你很有可能就是被形式而不是内容所驱动着，并且你正朝着模板僵尸的区域前进。类似地，如果开发流程仅仅因为模型（或者其他有用的东西）与模板不符，就阻止你把它们加入进来，你也许已经身陷于模板僵尸的区域了。当项目的对话集中于格式、布局、字体和编号系统上时，模板僵尸正蹒跚着从黑暗中冲你而来。

“只要参与过一两个软件项目，就一定会对书中介绍的情景深有感触，也必然会从中得到经验教训。”

——Joel Spolsky，《软件随想录》作者

“作者以十足的幽默感和深刻的洞察力写就此书。本书清楚地讲述了项目因何而失败，有何补救措施，并以非常友善和令人乐于接受的方式提出了切实可行的建议。”

——Warren McFarland，哈佛商学院教授

“对于任何一位曾经在组织里面从事过项目工作的人来说，86个项目模式熟悉得令人心惊。幸运的是，其中有一些模式是良性的，应该给予鼓励。然而悲哀的是，剩下的绝大多数模式不仅仅令人心灰意冷，而且它们对生产率、质量和项目团队士气的破坏程度令人瞠目结舌。”

——Ed Yourdon，《死亡之旅》作者

“这本《项目百态》就是关于项目管理的实话集……读这样一本书，你会笑，更多的时候你会摇头苦笑，甚至如芒在背。”

——熊节，ThoughtWorks中国公司首席咨询师，《重构》译者



图灵网站：www.turingbook.com 热线：(010)51095186
反馈/投稿/推荐信箱：contact@turingbook.com
有奖勘误：debug@turingbook.com

分类建议 计算机/IT项目管理

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-24488-8

定价：39.00元